ARTE DIGITAL: UNA PROPUESTA DE INSTALACIÓN BASADA EN TÉCNICAS DE PERCEPCIÓN Y VIDA ARTIFICIAL MEDIANTE LA CREACIÓN DE UN SDK PARA INSTALACIONES GENÉRICO

Antonio José Sánchez López

Facultad de Informática.
Universidad de Las Palmas de G.C.

Facultad de Informática. Universidad de Las Palmas de G.C.

Proyecto fin de carrera

Título: Arte Digital: Una Propuesta de Instalación basada en Percepción y Vida Artificial
Apellidos y nombre del alumno: Sánchez López, Antonio José
Fecha : Octubre 2011

Tutor: Castrillón Santana, Modesto

Facultad de Informática. Universidad de Las Palmas de G.C.

Agradecimientos

Aquí ponemos los agradecimientos

Facultad de Informática. Universidad de Las Palmas de G.C.

Índice general

Facultad de Informática. Universidad de Las Palmas de G.C.

Capítulo 1

Introducción

La introducción es lo primero que se lee, pero habitualmente lo último que se escribe. Pues su redacción depende de cómo se hayan escrito todas las otras secciones. Normalmente la introducción incluye una descripción muy general del proyecto y termina con un desglose del contenido de la memoria.

A lo largo de la historia, diversas disciplinas han hecho uso de la representación bidimensional para mostrar propuestas creativas. El ordenador es en la actualidad una herramienta de enorme potencial para el arte visual [12], tanto en el marco de la imagen estática, como en el contexto donde el factor tiempo se introduce en el proceso expresivo.

Las imágenes son fácilmente comprendidas por los humanos, motivo por el cual es un ámbito válido de trabajo creativo. Por otro lado, esa diversidad posible en una imagen ocupa también a multitud de científicos del campo de la Visión por Computador en su búsqueda de técnicas para detectar y reconocer objetos en ese espacio de representación: la imagen. Espacio donde un humano reconoce objetos conocidos con gran facilidad.

Desde el punto de vista de la imagen estática, una imagen es una matriz de píxeles que representan un punto en un espacio de muy alta dimensionalidad. La tecnología digital en este contexto, presenta la singularidad de la no existencia de un original único, el arte digital permite disponer del original en cualquier parte, éste es copiable hasta la saciedad sin pérdida. Adicionalmente, nuevos ámbitos tecnológicos han ido abriendo capacidades y posibilidades expresivas. Las tecnologías del vídeo digital y la animación introducen el factor tiempo en el proceso expresivo, la mutabilidad, la fugacidad y la narrativa temporal. La introducción de la interactividad a través del uso de las tecnologías de visión por computador aporta un nuevo canal expresivo y unas posibilidades para la generación de sensaciones a través de los conceptos de obra viva e interactiva, tal y como ya describiera Krueger en su concepto de Realidad Artificial [9].

La obra se puede convertir así en única y cambiante, reactiva a la interacción con su entorno en cada momento. Recupera el concepto de exclusividad, siendo además posible registrar la vida de la instalación. Este enfoque se relaciona de forma clara con el concepto de instalación manejado en el mundo artístico una obra es instalación si dialoga con el espacio que la circunda [8].

La motivación de este proyecto es investigar el uso de capacidades actuales de Visión por Computador y Vida Artificial para su integración en instalaciones artísticas. Hay que destacar que nuestra experiencia se relaciona fundamentalmente con el mundo tecnológico, por tanto, no es nuestro objetivo presentar una obra de creación, sino mostrar las posibilidades interactivas que la Inteligencia Artificial puede introducir en el arte, yendo más allá de aplicaciones básicas en visión de segmentación de figura y fondo en contextos restringidos como en Messa di voce [10], y abordando el contexto de detección de personas y vida artificial.

La exploración puede plantear y profundizar en nuevas posibilidades de interfaces y formas de interacción hombre-máquina.

Abordar cuando tenga paciencia para repasar documentación y referencias e inspiración para escribir bonito...

Capítulo 2

Estado actual del tema

Descripción del estado actual del tema con referencia a trabajos anteriores en el caso de proyectos que sean continuación o relacionados con otros proyectos.

Abordar cuando tenga paciencia para repasar documentación y referencias e inspiración para escribir bonito...

Capítulo 3

Metodología

Metodología a utilizar para el desarrollo del proyecto, herramientas de análisis, etc..
Las Metodologías de Desarrollo Ágiles son un marco de trabajo conceptual de la ingeniería del software que propone realizar iteraciones de las distintas fases a lo largo del desarrollo. Cada metodología introduce sus propias definiciones, pero en conjunto reflejan siguientes fases: Planificación, Análisis de Requerimientos, Diseño, Codificación, Revisión y Documentación. Una iteración no agrega demasiada funcionalidad en relación al producto final, pero permite tener una demo de la misma al final de cada iteración, momento en el que se evalúa de nuevo las prioridades del proyecto. Así se consigue un producto que podrá probarse desde las primeras semanas al que ir añadiendo funcionalidades [27].

La fortaleza del desarrollo ágil se centra en minimizar los riesgos desarrollando software en lapsos cortos y en su capacidad de respuesta al cambio, enfatizando el software funcional como objetivo y promoviendo las comunicaciones eficientes. [26]

En nuestro caso concreto se ha decidido utilizar como metodología ágil el Proceso Unificado Esencial (Essential Unified Process - EssUP), creado por Invar Jacobson [28]. EssUp plantea un conjunto de nueve prácticas ligeras y compatibles, que pueden adoptarse de forma individual o combinadas según las necesidades de un proyecto en cuestión. Estas prácticas son: Definición de Casos de Uso, Definición de Iteración, Definición de Arquitectura, Definición de Componente, Definición de Modelo, Definición de Producto, Definición de Proceso, Definición de Equipo y Definición del Proceso Unificado.

Para el desarrollo de la aplicación se hará uso de los apartados necesarios en cada fase del proyecto. Invar Jacobson también ofrece software enfocado al seguimiento de estas tareas de forma gratuita. Estas herramientas son:

- EssWork: Herramienta para el desarrollo de software orientado a la práctica que combina tareas para un trabajo flexible, con coherencia y efectivo.

- Essential Modeler: Herramienta visual que permite crear diagramas de casos de uso UML y modelos de clases.

Finalmente, para el caso del Diseño de la Base de Datos cabe destacar que se usará el modelo de Diagramas de Entidad-Interrelación, que se continuará con un proceso de transformación mediante Diseño Relacional para obtener finalmente la Base de Datos.

Capítulo 4

Recursos necesarios

Se detallan los recursos hardware, software u otros necesarios para el desarrollo del proyecto, incluyendo desde las aplicaciones necesarias así como librerías a utilizar y requerimientos hardware.

4.0.1. Software

- Aplicaciones:
  - TexMaker: Editor de LaTex.
  - EssWork: Editor de modelos de Casos de Uso.
  - Essential Modeler: Para el seguimiento y orientación de la metodología EssUp.
  - Dia: Para modelado de diagramas UML y Entidad-Interrelación.
  - Visual Studio 2008 Express Edition: Entorno de desarrollo.
  - Blender: Programa de edición 3D.
  - Gimp: Programa de edición 2D.
  - HeidiSQL: Front-end de MySQL.
- Librerías:
  - De Visión por Computador: OpenCV.
  - Motor gráfico: Se estudiarán UDK, CryEngine 3 ESDK, Crystal Space, OpenSceneGraph, Ogre, Irrlicht y Panda3D.
  - Interfaz gráfica de usuario: se estudiarán Qt y wxWidgets.
  - Para captura de audio y reproducción 3D se estudiará las capacidades del motor gráfico o de juego a usar, así como Fmod, OpenAL, SDL Mixer, Clunk, Irrklang y SFML.
  - Para composición de audio: Se estudiarán Chuck y Marsyas.
  - De sistema: Boost y SFML.

  - BBDD: MySQL, PostGreSQL.
  - De documentación: Doxygen.

- Recursos:

  - Imágenes: GCTextures.
  - Sonidos: wiki.laptop.org,go,Soundsamples; Jamendo.

4.0.2. Hardware

- Primera aproximación:

  - Entorno controlado: fondo estático, buena iluminación, sin ruido ambiente.
  - 1 PC con: 1 Monitor/Proyector, 1 cámara, micrófono y altavoces sonido envolvente.

- Segunda aproximación:

  - Entorno controlado: fondo estático, buena iluminación, sin ruido ambiente, mobiliario.
  - 1 PC con: 1 Monitor/Proyector, 2 cámaras, micrófono y altavoces sonido envolvente.

- -Tercera aproximación:

  - Entorno controlado: fondo estático, buena iluminación, sin ruido ambiente, mobiliario.
  - 1 PC con: 2 Monitores/Proyectores, 2 cámaras, micrófono y altavoces sonido envolvente.

- Cuarta aproximación:

  - Entorno controlado: fondo estático, buena iluminación, sin ruido ambiente, mobiliario.
  - Varios PCs con: 2 Monitores/Proyectores, 2 cámaras, micrófono y altavoces sonido envolvente.

Capítulo 5

Plan de trabajo y temporización

Desarrollo del plan de trabajo desglosado en etapas, con una estimación en cada etapa del tiempo de ejecución

- Etapa 1: Acercamiento
- Etapa 2: Primera Demo
- Etapa 3: Funcionalidades Añadidas
- Etapa 3+i: Funcionalidades Añadidas
- Etapa n: Revisión Final
- Etapa n+1: Publicidad

A continuación vendrán las secciones donde se desarrollan cada una de las etapas del proyecto

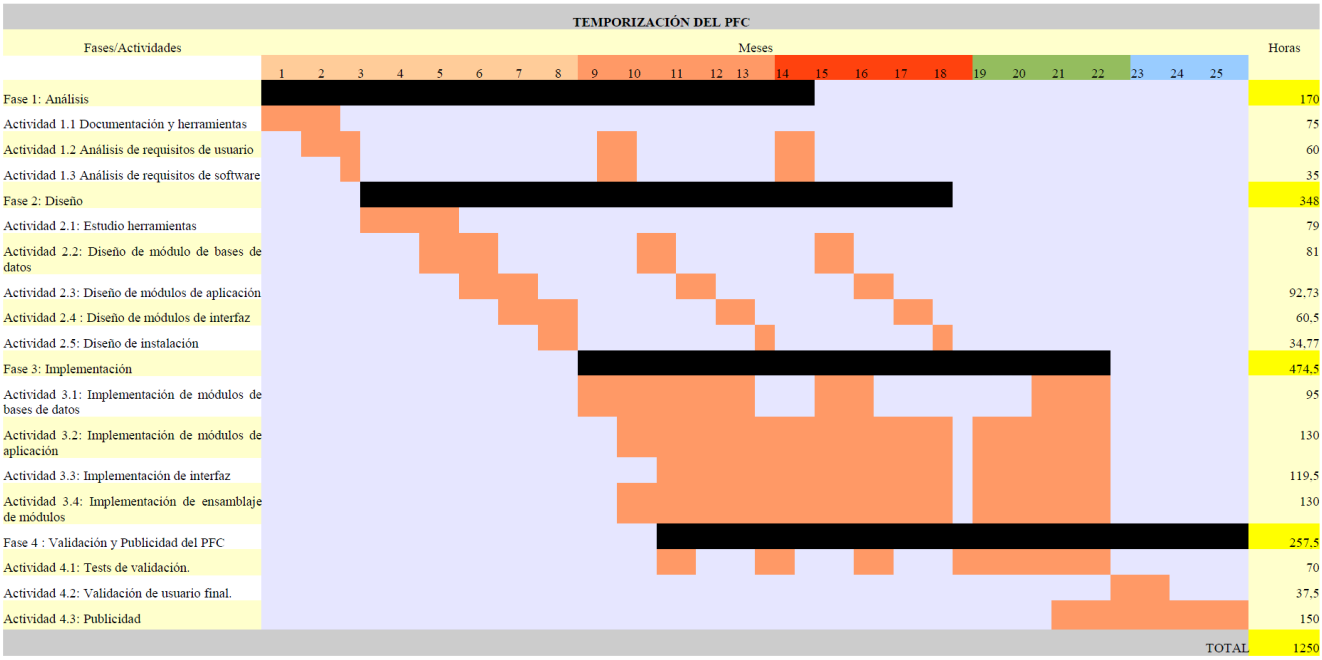


Figura 5.1: Plan de trabajo y temporización

Capítulo 6

Etapa 1: Acercamiento

En esta fase se llevará a cabo la primera aproximación a la solución que se va adoptar. Para ello se hará un estudio global de las capacidades de las que se desea dotar a la aplicación, que pretende ser completo aunque no en profundidad. También se hará una búsqueda y estudio de las herramientas disponibles que puedan ofrecer las funcionalidades necesarias para la construcción del proyecto.

En las siguientes fases/iteraciones se abordarán y trabajará en profundidad un subconjunto de cada apartado con el objetivo de conseguir demos intermedias que sean funcionales.

6.1. Análisis

6.1.1. Documentación y herramientas

Existe un conjunto de librerías bastante amplio que puede ser de utilidad en el desarrollo del proyecto. A continuación se hace una revisión de las mismas con el fin de elegir las que mejor se adapten a las necesidades del proyecto. Finalmente se elegirán las más adecuadas.

(RECORDAR AÑADIR LA FEATURE-LIST DE LOS DISTINTOS MOTORES - copy paste en forma de tablas bonitas)

UDK: Unreal Development Kit

A noviembre de 2009, el motor gráfico Unreal3, desarrollado por Epic Games, es considerado uno de los más potentes y de mayor calidad junto con el motor CryEngine de Crytek. A mediados del mismo mes Epic Games decide liberar UDK, su kit de desarrollo para proyectos sin ánimo de lucro, además de ofrecer licencias más asequibles para proyectos comerciales. Inicialmente el coste de una licencia para Unreal3 oscilaba alrededor los $700.000. Esta nueva situación hace posible tomarlo en consideración para el desarrollo de la aplicación que tratamos, por lo que se procede a su estudio. Es interesante comentar que Epic Games ya trabaja en la cuarta de versión de su motor, Unreal4, que planea estar disponible a partir de 2012-2018.

Algunas de las compañías que han usado Unreal 3 son: Atari, Activision, Capcom, Disney, Konami, Koei, 2K Games, Midway, THQ, Ubisoft, Sega, Sony, Electronic Arts, Square Enix y 3D Realms. Entre los ejemplos más destacados se encuentran las series BioShock o Gears of War, entre otros juegos como Medal of Honor: Airbone y Unreal Tournament 3.

Sin duda, los resultados que pueden verse en las imágenes corresponden a proyectos de grandes compañías que disponen de una importante cantidad de recursos. Como muestra visual más razonable se encuentra uno de los juegos de ejemplo que puede encontrarse en la página oficial del UDK, Whizzle.

Al ser un kit de desarrollo completo su lista de capacidades es muy extensa. En la siguiente tabla se muestra un resumen de su lista de capacidades de render:

Finalmente, se exponen algunas de las características clave para la toma de decisión:

- UDK es un kit de desarrollo software que ofrece un entorno completo para el desarrollo de videojuegos o aplicaciones similares.
- Está implementado con C++.
- Aunque el motor Unreal3 es multiplataforma, UDK sólo está disponible para los Sistemas Operativos Windows por ahora.
- El desarrollo de una aplicación en UDK puede realizarse en gran medida de forma visual a través de las opciones de la interfaz del entorno o escribiendo código en el lenguaje propio UnrealScript, similar a C++.
- El código escrito en UnrealScript se ejecuta sobre una máquina virtual propia.
- La máquina virtual de UnrealScript simula ejecución multihilos. Destacan que permiten la gestión de grandes cantidades de hilos, cantidad que hecho de otra forma, con el sistema de hilos nativo de Windows, podría provocar problemas.
- La ejecución de código UnrealScript es mucho más lento que C++. Detallan que el código en C++ es 20x más rápido.
- No se tiene acceso al código fuente del motor.
- La interacción con otro tipo de código en tiempo de ejecución debe hacerse a través de ficheros (.ini) que pueda leer UnrealScript, modificando o implementando controladores de periféricos, o a través de conexiones TCP/UDP. Este punto dificulta en gran medida la interacción con otras librerías necesarias para el desarrollo del proyecto.
- Es gratis para uso no comercial. En proyectos comerciales es necesario abonar una licencia y un porcentaje de las ventas en concepto de royalties que puede llegar al 25 % para ingresos superiores a 5000 euros.

Las capacidades del motor gráfico son sobresalientes. Sin embargo, el esquema de desarrollo con UDK no se ajusta adecuadamente al del resto de la aplicación.

CryENGINE 3 Educational SDK

Como era de esperar, poco después de la salida del Unreal Development Kit por parte de Epic Games, otro de los grandes motores gráficos comerciales lanzó su propia alternativa. Se trata en este caso del motor CryENGINE 3, de Crytek, con su propio kit de desarrollo con licencia gratuita. Sin embargo, hay que destacar que los términos de la misma son mucho menos flexibles que la de Epic Games y enfocado sólo hacia un sector. El SDK sólo está disponible para instituciones académicas por parte de jefes de curso, exclusivamente para proyectos de naturaleza interna y con fines no comerciales. Se excluye específicamente el caso de estudiantes individuales o proyectos de grupo. Además, los términos de la licencia en sí no son públicos y sólo son accesibles para personal académico previa solicitud.

Por ahora el único juego que parece usar este motor es Crysis 2, actualmente en desarrollo por Crytek.

Aunque una alternativa más limitada, en términos de licencia de uso, y en contradicción con algunos puntos de la naturaleza del proyecto, como es el uso de software gratuito y en la medida de lo posible libre, podría ser una alternativa a tener en cuenta y se procede a su análisis.

En la siguiente tabla se muestra un resumen de su lista de capacidades de render:

Finalmente, se exponen algunas de las características clave para la toma de decisión:

- Licencia de uso muy restrictiva. Sólo para proyectos académicos no comerciales y no puede ser solicitada por estudiantes individualmente. Continuidad del proyecto también limitada.

Crystal Space

Crystal Space es un entorno de desarrollo comúnmente usado con motor de juego. Está escrito en C++ usando un modelo Orientado a Objetos y soporta las principales plataformas. El SDK se distribuye bajo licencia LGPL, por lo que puede utilizarse de forma gratuita para cualquier tipo de desarrollo. Durante mucho tiempo se le ha considerado como uno de los motores libres más completos y populares, ya que no sólo es un motor de render sino un motor de juego, que incluye distintos módulos que facilitan el desarrollo.

Algunos ejemplos de uso son PlaneShift, El Hierro Virtual (ULPGC) y el reciente Yo Frankie! (Apricot Open Game Project).

- Relación estrecha con Blender, entorno de diseño 3D y de desarrollo de videojuegos con el que se dispone de cierta experiencia.

- Complejidad alta y relativamente abandonado, pasando largos períodos de tiempo entre revisiones.
- Nueva versión del SDK publicada el 25 de Enero de 2010, que incluye algunas mejoras y pocas funcionalidades añadidas. Entre ellas destaca el uso de un plugin basado en OpenAL para ofrecer sonido 3D.
- Resultados visualmente pobres y de bajo rendimiento incluso en proyectos desarrollados por equipos con experiencia.

(completar)

Ogre, Object-Oriented Graphics Rendering Engine

Ogre es considerado uno de los motores gráficos open-source multiplataforma más populares y de más éxito. Está escrito en C++ y está dirigido al mismo lenguaje, aunque existen wrappers para Python, Java y .NET.

A diferencia de otros, no está planteado como un motor de juego, sólo como motor gráfico. Aunque se le puede añadir con facilidad otras librerías. La filosofía de Ogre es que el desarrollador pueda añadir Ogre como un módulo más en su aplicación, así como los módulos que necesite, en lugar de tener que adaptar la aplicación para que encaje correctamente con una solución que pretenda abarcar más de lo que se necesita y provocando incomodidades e incompatibilidades.

- Altamente modular, se le pueden añadir librerías y plugins con facilidad.
- Esto mismo hace que las capacidades sean más limitadas que las de los motores y SDKs.
- Dispone de un equipo de desarrollo dinámico y de una comunidad muy activa.
- Nueva versión publicada el 31 de Diciembre de 2009, con una lista amplia de mejoras.
- Utiliza la licencia del MIT-X11, por lo que Ogre es gratuito, libre y puede usarse para cualquier tipo de desarrollo.

(completar)

OSG: OpenSceneGraph

OSG se presenta como una librería multiplataforma para el desarrollo de aplicaciones que requieran de visualización 3D con alto rendimiento, como visualización científica, realidad virtual o comúnmente juegos.

Es una librería ampliamente usada, con cierta tendencia hacia la visualización científica y geográfica, con una lista amplia de capacidades. Algunos ejemplos de uso son TerrainView, Pok3D, Nasa'a Blue Marble y Capaware!/Geviemer (ULPGC)

- Altamente modular y potente, con multitud de extensiones.
- Dispone de un equipo de desarrollo dinámico, con revisiones frecuentes y de una comunidad activa.
- Se dispone de cierta experiencia con el entorno.
- El diseño de la API puede ser confuso y en cierta medida deficiente en términos de facilidad de uso.
- Documentación pobre.
- Utiliza licencia LGPL, por lo que es gratuita, libre y puede usarse para cualquier tipo de desarrollo.

(completar)

Irrlicht: Lightning Fast Realtime 3D Engine

Motor gráfico 3D multiplataforma de código abierto y de alto rendimiento. Está escrito en C++, pero también esta disponible para lenguajes .NET, así como otros lenguajes como Java, Perl, Ruby, Python o Lua, mediante bindings.

- Altamente modular, potente y con multitud de extensiones.
- Dispone de un equipo de desarrollo dinámico, con revisiones frecuentes y de una comunidad activa.
- Utiliza la licencia zlib/libpng, por lo que es gratuito, libre y puede usarse para cualquier tipo de desarrollo.
- Mediante IrrEDIT se peude construir escenarios con facilidad que peuden usarse dentro de Irrlycht.
- Sin embargo, otras librerías como IrrKlang, para sonido, se distribuyen bajo licencias más restrictivas.
- Se define a la altura de los motores comerciales pero tiene ciertas carencias.

(completar)

Panda3D: Free 3D Game Engine

Originalmente Panda3D, acrónimo de Platform Agnostic Networked Display Architecture, fue creado por Disney como parte de una de sus atracciones y posteriormente liberado, en 2002, con la intención de facilitar el trabajo con universidades y projectos de investigación en Realidad Virtual.

Acogido por el Carnegie Mellon Entertainment Technology Center, fue mejorado y preparado para su uso público. Panda3D se define como un motor de juego y un entorno de de trabajo para render 3D y desarrollo de juegos, libre, gratuito y multiplataforma, que puede usarse para cualquier tipo de desarrollo. Incluye motor gráfico, audio, gestión de entrada/salida y detección de colisiones entre otras capacidades.

- Dispone de un equipo de desarrollo dinámico, con revisiones frecuentes y de una comunidad activa. Aunque mayormente para Python.
- Sin embargo, existe un menor dinamismo y soporte para C++ que para Python.
- Tiene un diseño orientado a mejoras futuras e incluye funcionalidades avanzadas cercanas a los motores comerciales que aún no incluyen otros motores libres.
- Diseño de la API atractivo y sencillo de usar.
- Énfasis en la documentación de la librería, en distintas modalidades: manuales, referencias, y plantillas conceptuales, que facilitan en gran medida la comprensión y el uso de la librería.
- Se añade a la documentación tradicional diversos video tutoriales creados tanto por la comunidad, así como clases impartidas sobre Panda3D por David Rose, del Instituto de Realidad Virtual de Walt Disney.
- Ampliamente usada por alumnos del Carnegie Mellon y en proyectos de equipos pequeños con resultados visuales atractivos y con buen rendimiento.
- Se distribuye bajo una licencia basada en BSD que permite su uso libre y gratuito para todo tipo de proyectos.

(completar)

Doxygen: Source code documentation generator tool

Generador de documentación multiplataforma para C++, entre otros lenguajes. Permite generar documentación online para su visualización en un navegador web (HTML), así como LaTex, MS-Word, PostCript, PDF con hipervínculos entre otros. Además puede usarse para extraer la estructura de código de fuentes no documentados, y generar grafos de dependencias, diagramas de clases y otros esquemas.

Doxygen se distribuye bajo licencia GPL. De cualquier forma, los documentos producidos mediante Doxigen son trabajo derivado de los datos usados en su producción, por lo que no se ven

afectados por la licencia.

No se estudian más casos por considerarse la mejor(¿única?) opción y tener cierta experiencia con ella, lo que es una ventaja determinante.

Boost C++ Libraries

Boost es un conjunto de librerías de código abierto multiplataforma con la intención de extender las capacidades del lenguaje de programación C++. Varios fundadores de Boost forman parte del Comité ISO de Estándares de C++, por lo que algunas de estas librerias terminan por introducirse en la siguiente versión estándar de C++. Utiliza una licencia propia, la Boost Software License, que permite su uso en cualquier tipo de proyectos, comerciales o no.

Algunas de las librerías de mayor uso son las que facilitan las operaciones con el sistema, I/O y gestión de hilos, entre muchas otras.

Qt: Cross-platform Application and UI Framework

Librería multiplataforma de Trolltech para desarrollar interfaces gráficas de usuario, así como para desarrollar aplicaciones de consola y servidores. Utiliza como lenguaje C++, aunque también está disponible para otros lenguajes a través de binding (Python, C#, Ruby, Java, Ada y Php, entre otros).

Qt es ampliamente usado y considerado como una de las opciones multiplataforma más completa y estable. Algunos ejemplos de uso son principalmente KDE, a partir del cual logró un considerable éxito y expansión, además de otras aplicaciones como Google Earth o Skype.

Tras polémicas en sus inicios por publicitarse como código libre sin serlo, actualmente la biblioteca es gratuita y libre tomando en consideración las condiciones de las opciones LGPL 2.1 y GPL 3.0 para el proyecto. También existe otra opción de pago destinada para software comercial que no quiera cumplir con las condiciones anteriores.

wxWidgets: Cross-platform GUI Library

Se trata de una librería para C++ para el desarrollo de interfaces gráficas de usuario. Es una de las pocas opciones realmente multiplataforma, gratuita y open source disponibles. Tiene la capacidad para ofrecer interfaces de aspecto nativo dependiendo de la máquina sobre la que se ejecuta el código.

wxWidgets, aunque con sus desventajas y limitaciones, es una libería ampliamente usada que ha creado comunidad. Algunos de sus usarios más conocidos son: AOL(AOL Comunicator), California Institute of Technology (Gambit), Carnegie Mellon University (Audacity), Grisoft Inc.

(AVG Antivirus), NASA (NASGRO), National Human Genome Research Institute - USA (ComboScreen), TomTom (TomTom HOME), Xerox (VIPP) o la propia Universidad de Las Palmas de Gran Canaria (Capaware!) entre otros.

Se exponen algunas de las características clave para la toma de decisión sobre su uso:

- Librería multiplataforma gratuita y open source para desarrollar interfaces gráficas de usuario para aplicaciones de escritorio.
- No todas las opciones que ofrece la librería funciona en las distintas plataformas, por lo que hay que tener especial cuidado si que quiere que el proyecto sea, en potencia, multiplataforma. Esto limita las opciones de la librería que pueden usarse y la calidad visual del resultado.
- Existe documentación, foros y una comunidad bastante amplia y activa.
- Hay que tener en cuenta que durante el funcionamiento de la aplicación no existe una interfaz gráfica destinada al usuario en términos tradicionales. Es decir, no existen ventanas, botones, menús ni indicadores destinados a que el usuario las maneje. Por lo tanto no es un apartado crítico del proyecto. Sin embargo, se desea una interfaz gráfica que permita preparar la instalación en su situación final, además de gestionar ventanas y para mostrar las opciones de configuración de la aplicación. Las capacidades de wxWidgets resultan suficientes para estas necesidades.
- Se dispone de amplia experiencia previa con la librería en otros proyectos.

Se decide optar por esta librería por cubrir las necesidades básicas del proyecto, por ser gratuita y libre, y por tener una extensa experiencia con la misma en otros proyectos, lo que es determinante para considerarla la mejor opción.

OpenCV: Open Source Computer Vision

OpenCV es una librería multiplataforma de funciones de visión por computador en tiempo real. Está desarrollada inicialmente por Intel, siendo gratuita tanto para uso comercial como investigación bajo licencia BSD. Surgió en 1999 como una iniciativa de Intel para mejorar aplicaciones intensivas en CPU, formando parte de una serie de proyectos que incluían ray tracing en tiempo real y pantallas de representación 3D. Actualmente acaba de salir la versión 2.0 que incluye amplias mejoras a la interfaz con C++, mejor prototipado, nuevas funciones y mejoras de rendimiento, especialmente en sistemas multicore.

Algunas de sus aplicaciones son HCI (Human-Computer Interaction), Identificación, Segmentación y Reconocimiento de objetos, Reconocimiento de Caras, Reconocimiento de Gestos, Motion tracking, Ego Motion, Motion Understanding, SFM(Structure from Motion), Calibración estéreo y multi-cámara, Percepción de Profundidad y Robótica móvil.

La librería es utilizada en el proyecto principalmente para las tareas de reconocimiento de objetos, caras, gestos y movimiento, lo que formaría parte del Módulo de Reconocimiento. Además,

también es utilizada para el proceso de imágenes con fines estéticos, adoptando los resultados de los algoritmos de reconocimiento como modificadores de las imágenes en si, formando parte del Módulo de Producción.

Fmod: Music & Sound Effects System

Librería de audio propietaria y multiplataforma de Firelight Technologies que soporta un amplio abanico de formatos de audio. También tiene capacidad para reproducir sonido 3D en sistemas de sonido envolventes.

La biblioteca es ampliamente utilizada en juegos y varios motores gráficos incluyen soporte para la misma. Algunos ejemplos son: BioShock, Call of Duty 4, Crysis, Far Cry, la saga Guitar Hero, Heavenly Sword, Hellgate: London, Metroid Prime 3, Second Life o World of Warcraft, entre otros.

Fmod está disponible siguiendo distintos esquemas. Sin embargo, no es de código abierto y sólo es gratuito para desarrollo de aplicaciones no comerciales.

OpenAL: Cross-platform 3D Audio API

API de audio multiplataforma desarrollada por Creative Labs destinada la reproducción de audio posicional y multicanal en 3D. Se ideó para su uso extenso en videojuegos siguiendo las mismas convenciones que OpenGL, consiguiendo convertirse en un estándar aceptado.

La biblioteca es ampliamente usada en juegos y varios motores incluyen soporte para la misma. Algunos ejemplos relativamente recientes son: Doom3, Quake4, Unreal2, Unreal Tournament 3 o Hitman2.

Sin embargo, en los últimos años el mantenimiento de la misma se ha descuidado y aparecen errores, especialmente en sistemas de 64 bits. Son comunes las nuevas variaciones de OpenAL que le dan continuidad, como OpenAL Soft, OpenAL++ o distintas librerías a modo de wrapper que usan OpenAL por debajo, como SFML.

De cualquier forma, OpenAL parece ser la única opción multiplataforma gratuita y libre para cualquier tipo de desarrollo que provea de posicionamiento 3D de audio en sistemas de sonido envolventes.

SDL_Mixer: Simple DirectMedia Layer Mixer

Partiendo de la SDL, librería multiplataforma que permite el acceso de bajo nivel a dispositivos de audio, periféricos y hardware 3D, SDL_Mixer se centra en el primer apartado facilitando la mezcla de sonido multicanal, así como la carga de samples y de música de distintos formatos.

Sin embargo, aunque permite el acceso a bajo nivel, no aporta mayores funcionalidades por sí misma. Bla bla bla...
(completar)

Clunk: Open Source 3D-Sound library

Biblioteca de creación reciente de código abierto para C++ que pretende dar soporte para la generación de sonido 3D, binaural, en tiempo real. Propone una API bastante manejable y sencilla de usar, con un modelo orientado a la gestión de objetos. Sin embargo aún se encuentra en fase de testeo antes de lanzar su primera release.

Sin embargo, está preparado para generar sonido binaural, el cual tiene su mejor efecto en auriculares y no tanto en altavoces. Por otro lado del modelo de escucha sólo tiene en consideración posición, velocidad y dirección de orientación, sin información de verticalidad; por lo que no es posible definir realmente su estado en un entorno 3D, sólo en un plano.

Irrklang: High level 3D audio engine/API

API de alto nivel para sonido 2D/3D multiplataforma enfocado hacia C++ y lenguajes.NET. Da soporte para un amplio abanico de formatos de sonido y provee de una API muy sencilla de usar. Se puede encontrar bajo distintas licencias pero sólo es gratuito para desarrollos no comerciales.

Al igual que OpenAL y FMOD éste sí permite definir completamente las propiedades necesarias para orientar en 3D tanto los sonidos como la escucha. Sin embargo, la limitaciones de licencia son menos interesantes que las de una opción más abierta.
(completar)

SFML: Simple and Fast Multimedia Library

SFML se ofrece como una API open-source multimedia que provee mecanismos para la gestión del sistema, de gráficos, interfaz gráfica de usuario, sonido, periféricos y de red. La librería es multiplataforma y se encuentra disponible para varios lenguajes como C, C++,.NET, Python o Ruby, entre otros.

Sigue un diseño Orientado a Objetos y define un interfaz fácil de usar y de integrar en otros proyectos. Se compone de diferentes paquetes que pueden ser usados en conjunto o individualmente. Es de especial interés el paquete de sonido, que funciona a modo de wrapper de OpenAL, arreglando algunos de los bugs que tiene y simplificando su uso.

Se distribuye bajo licencia zlib/png, de la Open Source Initiative, por lo que es gratuita y abierta para todo tipo de proyectos.

Tiene varios defectos. Por un lado, en el paquete de sonido la documentación es errónea, y puede llevar a confusiones importantes. Además la definición de la API, aunque más simple, limita la funcionalidad real de OpenAL hasta el punto de que no puede orientarse correctamente la escucha en 3D, por lo que sólo puede usarse en 2D o 3D en el plano horizontal. Sin embargo, pequeñas modificaciones en la libería permitirían recuperar esa funcionalidad sin problemas.

Chuck: Strongly-timed, Concurrent, and On-the-fly Audio Programming Language

Chuck es un lenguaje de programación para el análisis, síntesis, composición y producción de audio. La librería es multiplataforma y presenta un modelo de programación concurrente basado en tiempo, altamente preciso (strongly-timed) y con la habilidad de añadir y moficar código en tiempo de ejecución. Chuck soporta dispositivos MIDI, OSC, HID y audio multicanal.

(completar, scripts, etc y estudiar más a fondo junto con Tapestrea, en comparación con Marsyas)

Marsyas: Music Analysis, Retrieval and Synthesis for Audio Signals

Librería y framework para C++ para el procesado de audio con especial énfasis en aplicaciones de extracción de información en música, creada principalmente por George Tzanetakis y desarrollada de forma abierta. Está enfocado hacia el prototipado rápido y la experimentación en el análisis y síntesis de audio, procurando un alto rendimiento.

Bla bla bla...

(completar... y estudiar más a fondo en comparación con Chuck)

PostgreSQL

Potente Bases de Datos objeto-relacional open source. Es considerada la BBDD open source más avanzada y potente, con más de 15 años de desarrollo. Está disponible para múltiples sistemas operativos Linux, UNIX y Windows y dispone de interfaces con múltiples lenguages, entre ellos C++.

bla bla bla

OODBMS: Object Oriented Data Base Management Systems

Se han estudiado múltiples variantes de Bases de Datos Orientadas a Objetos. Sin embargo, no ha sido posible encontrar ninguna que se adapte a los requisitos software del proyecto. Se resumen detalles que implican su descarte:

- EyeDB: Disponible para C++, potente y estable. Sólo está disponible para Linux.
- Metakit: Base de datos embebida limita su uso y crecimiento futuro del proyecto.
- db4o: Embebida, sólo para Java o c#.
- Orient: Versión para Windows sólo embebida. Mala documentación.

- NeoDatis: Sólo para Java o C#.
- Perst: Embebida.
- Odaba: Proyecto antiguo, API confusa.
- Matisse: Licencia gratuitia restrictiva, sólo para prueba de la librería.
- Jade6: Licencia restrictiva, sólo para prueba de la librería.
- Bifröst: Abandonado.
- Cerebrum: ??
- Frontier: Abandonado.
- Oviedo3: Abandonado.
- Thor: Proyecto del MIT, no disponible aún.
- MongoDB: No es orientada a objetos sino a documentos, mala eficiencia en altas cantidades de transacciones.

Debea: Database Access Library

Se trata de una librería en C++ que actúa como mapper entre el modelo orientado a objetos de la aplicación y el esquema de persistencia elegido (base de datos SQL, CSV o ficheros XML). En el caso de SQL, aunque encapsula la persistencia de los objetos incluyendo una API sencilla para la carga y almacenamiento de los mismos y sus relaciones, sigue siendo posible ejecutar comandos SQL si se desea realizar consultas elaboradas. Usa la licencia de wxWindows por lo que es gratuita y libre. También está disponible para Linux y Windows. Soporta de forma nativa bases de datos SQLite3 y PostgreSQL, además de ficheros csv y xml y ofrece una API sencilla y una documentación clara.

6.1.2. Análisis de Requisitos de Usuario

La instalación se presenta como un espacio en el que el usuario puede entrar y moverse. Dentro de ese espacio se encontrará con una o varias pantallas, cámaras, micrófonos y altavoces. A través de las cámaras y de los micrófonos se capturará información para su análisis y para la producción de efectos, y mediante las pantallas y los altavoces se mostrará un entorno. Es mediante la actitud e interacción del usuario con el espacio como el entorno virtual es generado. A su vez, este entorno será capaz de actuar y evolucionar por su cuenta dentro de un esquema de comportamiento, también definido mediante la interacción del usuario durante la sesión. Capacidades añadidas permitirían a un usuario entrar en el entorno de otro usuario. Aunque este apartado se dejará como trabajo futuro en función de la evolución del proyecto.

Teniendo en cuenta estos aspectos, se detallan los siguientes requisitos:

- Captura de imagen del usuario.
- Captura de sonido del usuario.
- Reconocimiento de su posición en el espacio, localización de cabeza/cuerpo, posturas o gestos.
- Reconocimiento de voz (sonidos, palabras) (en función de la evolución del proyecto)
- Crear entorno propio (crear, borrar, cargar escenarios).
- generación de esquema de comportamiento.
- Generación de efectos visuales.
- Generación de efectos de audio y composición musical.
- Generación de elementos 3D.
- Generación de elementos 2D.
- Generación de vida artificial.
- Interacción.
  - Con la escena:
    - Vista: Perspectiva Visual (asociación cámara/cabeza).
    - Oído: sonido estéreo ó 3D envolvente.
    - Provocar sonidos: ruidos, palmas, habla.
    - Movimiento dentro de la escena:
      - ◇ No estar.
      - ◇ Estar.
      - ◇ Quedarse quieto.
      - ◇ Moverse.
      - ◇ Contacto del usuario con los elementos de la escena (velocidad: estático, roce, golpe, gesto...)
      - ◇ Elementos: Crear, destruir, mover, provocar interacción/respuesta.
      - ◇ Definición de Esquema de Comportamiento/Psique: propia y de la escena, estudio del comportamiento e interacción que sentará las bases del estado del entorno (tipo de escena, iluminación, sonidos, efectos, tipos de elementos creados, comportamientos de los mismos).
  - Con otros usuarios: (en función de la evolución del proyecto)
    - Entrar en escenarios de otros usuarios (limitación de interacción según permisos: ver/tocar/crear/destruir/psique-empatía(poder modificar esquema de comportamiento del otro usuario o que se modifique el suyo)).

    - Ver y comunicarse con otro usuario: avatar visual, comunicación por audio.

- Alta capacidad de personalización del entorno resultante y unicidad del entorno.

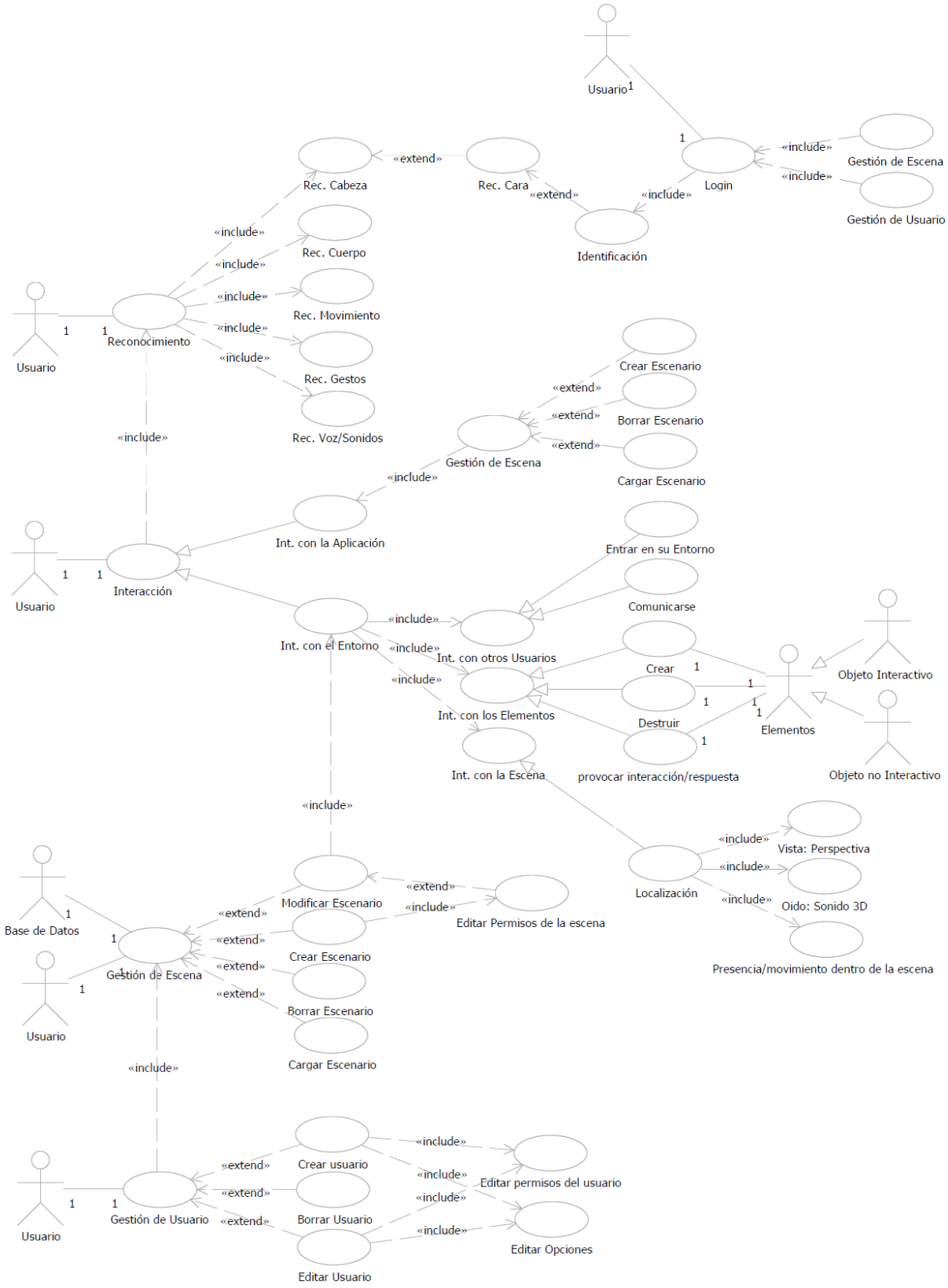


Figura 6.1: Modelo de Casos de Uso.

6.1.3. Análisis de Requisitos de Software

Además de conseguir como resultado una Instalación que muestre las capacidades tecnológicas en el entorno del Arte, así como la experimentación en nuestras interfaces y formas de interacción del usuario, este proyecto pretende ofrecer un entorno de trabajo para el futuro desarrollo de nuevas aplicaciones de este tipo. De esta forma, la aplicación se enfoca como un proyecto open-source, un marco de trabajo compuesto de diferentes módulos independientes, de forma que la implementación subyacente sea modificable sin afectar al resto de los módulos. Para ello se debe definir un completo esquema de interfaces.

Teniendo en cuenta estas características, se detallan los requisitos software del proyecto:

- Uso de software libre y multiplataforma.
- Uso de recursos gratuitos.
- El framework producido será software libre.
- Diseño del framework orientado a multiplataforma, para facilitar su futura portabilidad.
- Diseño de interfaces de forma que los módulos sean fácilmente sustituibles.
- Sistema local/online persistente (según evolución del proyecto).

6.2. Diseño

6.2.1. Estructuración de conceptos

(REVISAR Y DEFINIR BIEN)

Aplicación

-Aplicación: -Programa de que se ejecuta en un ordenador. -Carga un escenario para un usuario. -Puede entrar en escenarios cargados por otros usuarios en otras aplicaciones conectados al mismo sistema.

Usuario

-Usuario: -El usuario se presenta en la instalación y es reconocido por su cara (ID) y se crea un entorno/escenario. -El usuario puede querer crear un escenario nuevo o ver uno anterior (default: decidir cargar último/crear). -El usuario puede interactuar con un escenario propio o de otro usuario (preferencia, permisos) (default: todo, ver, tocar), and lógico bit a bit. -El usuario puede verse a sí mismo (avatar), a otros usuarios (avatar) y a otros escenarios en red (símbolos de escenarios). -Los cambios y la interacción del usuario define un patrón de comportamiento (psique, opciones: egocéntrico, tábula rasa).

-Permisos: -Conjunto de opciones que define la interacción entre un usuario(desea) y un escenario(permite). -Operación lógica (umpersand) definiría qué puede hacer un usuario en un escenario. -Posibles permisos: Ver/Tocar/Coger/Copiar/Crear/Destruir/Influenciar/LibreAlbedrío

Entorno

-Escenario/Entorno: -Se presenta ante / es creado por / un usuario. -Representa elementos vivos o inertes, definidos mediante objetos 3D, 2D, sonidos y efectos. -Responde ante la interacción del usuario y cambia según sus acciones. -También cambia por su cuenta. -Permite al usuario propietario interactuar cómo desee, la interacción de otros usuarios puede estar limitada. -Los cambios y la interacción del escenario define un patrón de comportamiento que establece las bases del mismo (psique). -Se rige por la psique pero tiene un determinado factor de aleatoriedad. -Componentes: Espacio, entidades.

-Espacio: -Terrenos (discutir estático/dinámico, interactivo o no). -Sonido ambiental: autogenerado, gramática para generación de composiciones.

-Entidades: -Usuarios: avatares. -Objetos no interactivos/inertes (estudiar interacción por físicas). -Objetos interactivos: vivos/inertes. -Sonido de las entidades: acciones (gramática), etc. (autogenerado: gramática). -Sonidos de los usuarios: acciones (autogenerado: gramática), captura por micrófono, comunicaciones chat, etc.

-Objetos interactivos: -Colección de elementos 3D, 2D, sonidos y efectos. -Composición: algoritmo genético que define la creación de un objeto, tanto física (3D, 2D, sonidos, efectos) como de comportamiento. -Capacidades de evolución: reproducción, mutación y asimilación. -Vivos: Agentes, revisar modelo BDI (Creencias, Deseos, Intenciones), y de Agentes Híbridos (Nivel Reactivo, Conocimiento, Social). -No-muertos: Agentes interactivos totalmente reactivos, sin deseos ni intenciones.

Resumen

Llegados a este punto se observan las siguientes necesidades:

- Aplicación que implemente y englobe las funcionalidades.
- Interfaz gráfica de usuario genérica para la gestión de ventanas y configuración de opciones de aplicación.
- Modificación de elementos y datos, carga y guardado de los mismos.
- Base de Datos.
- Percepción del entorno mediante técnicas de Visión por Computador y Captura de audio.
- Generación de elementos con los que componer un entorno (elementos 3D, 2D, audio).
- Análisis y definición del patrón de interacción del usuario y de la evolución del entorno (psique).

6.2.2. Diseño de la Aplicación

A partir de esta descripción se pueden diferenciar y extraer los diferentes módulos de los que es necesario que se componga la aplicación. Hay que tener en cuenta además que se desea conseguir una alta modularidad e independencia entre las distintas secciones, que permita extraer e intercambiar módulos con facilidad, p.e para usen distintas librerías o no usen librerías externas en absoluto. Por ello, se sigue un modelo de interfaces, en la que se define el núcleo de la aplicación y las relaciones entre los módulos. Sobre ella, se implementará cada módulo y finalmente se construirá la aplicación final.

- Core: Conjunto de interfaces de la aplicación.
  - IAplication: Define la interfaz de la aplicación completa. Es además una composición de los distintos módulos de la misma.
  - IGUI: Interfaz gráfica de la aplicación de ventanas, para las opciones básicas de visualización y configuración.
  - IPersistence: Persistencia de la aplicación, encapsula los cambios que se efectúan sobre los datos de la misma.
  - IPercept: Engloba la interfaz de usuario mediante percepción (Visión por Computador, captura de audio).
  - Iprod: Es el módulo referido a la Producción, generación de elementos 3D, 2D, composición de audio, etc.
  - ICog: Para el ánalisis de la interacción del usuario y definición del concepto de Esquema de Comportamiento/Psique de Entorno y de Usuario.
- VOX: Implementación de la aplicación.
- Monitor: Para la gestión y monitorización de los accesos a la Base de Datos.
- Base de Datos

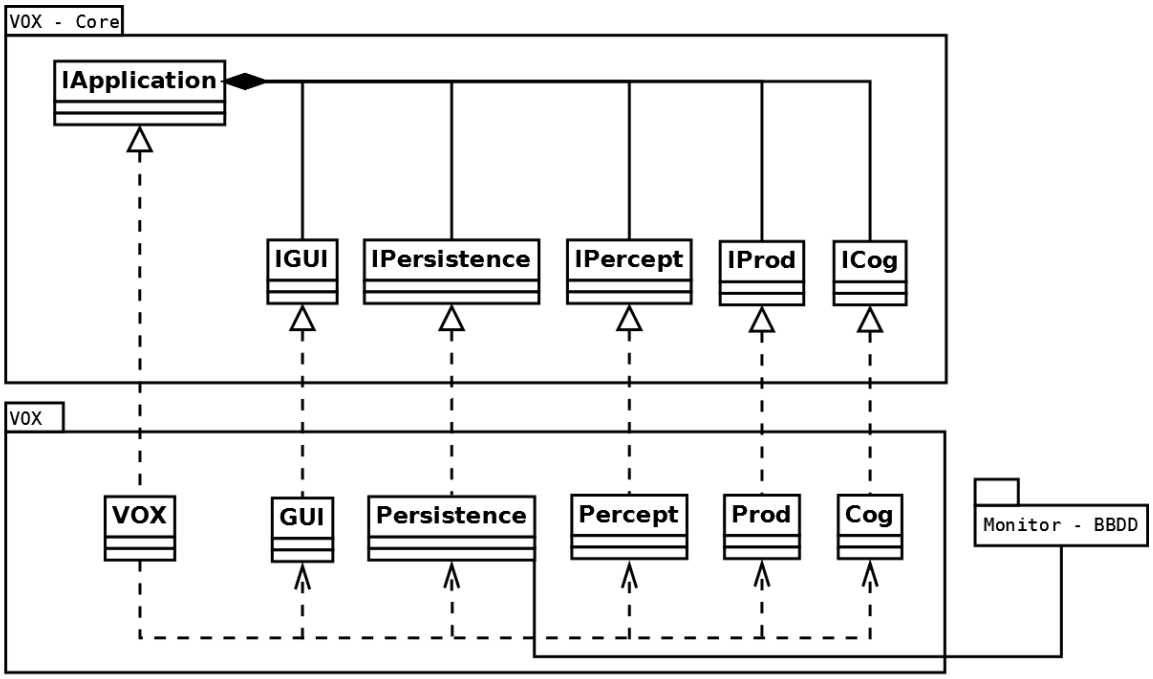


Figura 6.2: Diagrama de Clases UML.

Capítulo 7

Etapa 2: Aplicación principal, GUI, Percepción y Producción

Para el desarrollo de la primera demo se pretende conseguir una aplicación ejecutable que integre el modelo de interfaz y la separación de módulos definida en la etapa anterior, centrándose en implementar las funcionalidades de un subconjunto de módulos básicos.

Los módulos a abordar son aquellos que aporten la tecnología necesaria para que ésta se ejecute y provea de las herramientas necesarias para posteriormente analizar y generar contenido. Estos es, para capturar y mostrar la información que se manejará en la aplicación. Los módulos son: IGUI, IPercept e IProd.

La intención es únicamente crear la estructura básica de la aplicación, así como establecer una relación básica de dependencias entre módulos, cargando en cada caso correctamente las librerías utilizadas.

En este punto no existen los conceptos de usuario ni de entorno. Tampoco se abordará la Persistencia ni, por tanto, la Base de Datos.

7.1. Análisis

7.1.1. Análisis de Requisitos de Usuario

Como se ha comentado estudiaremos un subconjunto básico de los casos de uso para esta primera demo ejecutable. En ella se abordarán las funcionalidades básicas de gestión de aplicación y ventanas que permita su ejecución y mostrar información en distintas ventanas. Además se cargarán los módulos de percepción y de producción. Se mostrarán las imágenes capturadas por las cámaras y el audio grabado, así como un entorno 3D por defecto, con sonido posicional.

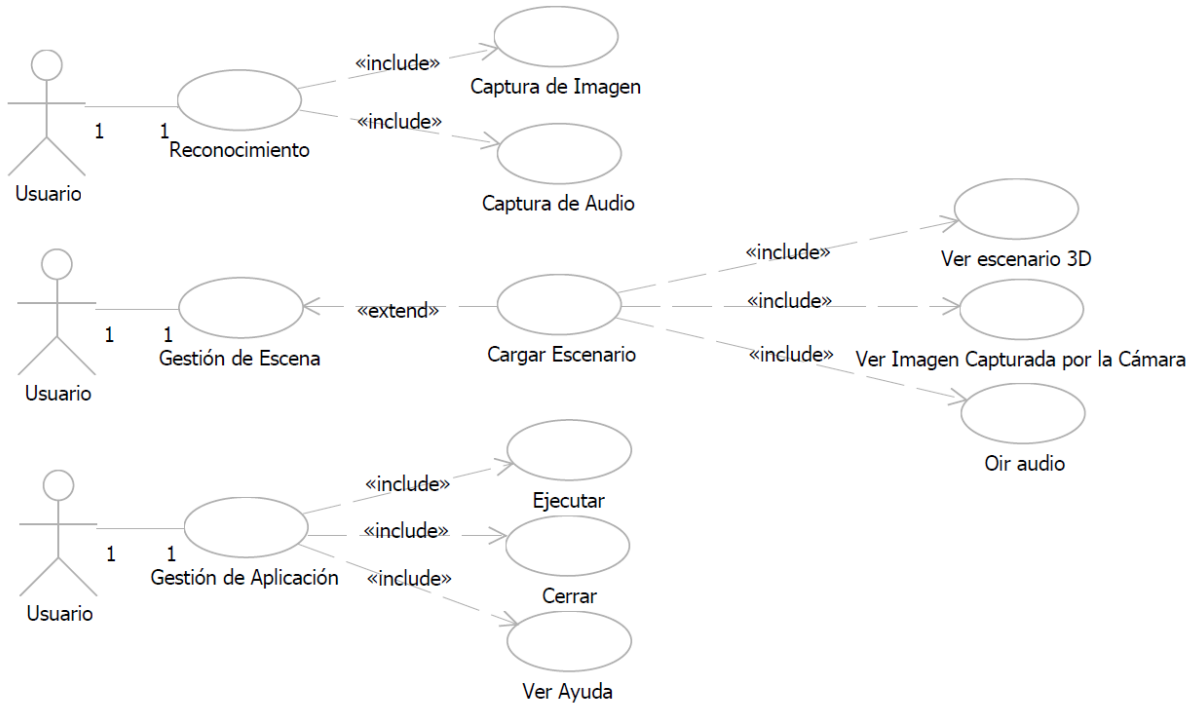


Figura 7.1: Modelo de Casos de Uso.

7.1.2. Selección de Herramientas

A partir del estudio de las herramientas disponibles se han seleccionado las más adecuadas, tanto por sus características como por compatibilidades técnicas en el conjunto del proyecto.

- GUI: wxWidgets
- Motor gráfico: Panda3D
- Gestión de Hilos: Boost
- Captura de Audio y Audio Posicional: SFML
- Captura de Video y Visión por Computador: OpenCV

7.2. Diseño

7.2.1. Diseño de la Aplicación

Breve descripción de los módulos

- core:
  - ICog: Interfaz básica para la creación del módulo.
  - IGui: Interfaz básica para la creación del módulo y registro de ventanas.
    - IGuiWindow: Creación, Mostrar/Ocultar.
  - IPercept: Interfaz básica para la creación e inicialización del módulo.
  - IPersistence: Interfaz básica para la creación del módulo.
  - IProd: Interfaz básica para la creación, intervención e inicialización del módulo.
  - Application: Interfaz básica para la creación de la aplicación.
- igui:
  - Crear ventana de aplicación.
  - Operación básicas de ventana: mostrar, mover, cerrar, cambiar contenido.
  - Menú de Aplicación: Archivo, Vista, Herramientas, Ayuda.
- ipercept:
  - Capacidad para capturar imágenes de n-cámaras.
  - Capturar imágenes de las n-cámaras.
  - Mostrar imágenes capturadas.
  - Capturar Audio.
- iprod:
  - Cargar una escena 3D por defecto.
  - Introducir y reproducir audio posicional 3D.
  - Capacidad para mostrar n-ventanas de render de la misma escena 3D, con vistas independientes.
- vox: Carga y ejecución de los distintos módulos.

Diseño General en UML

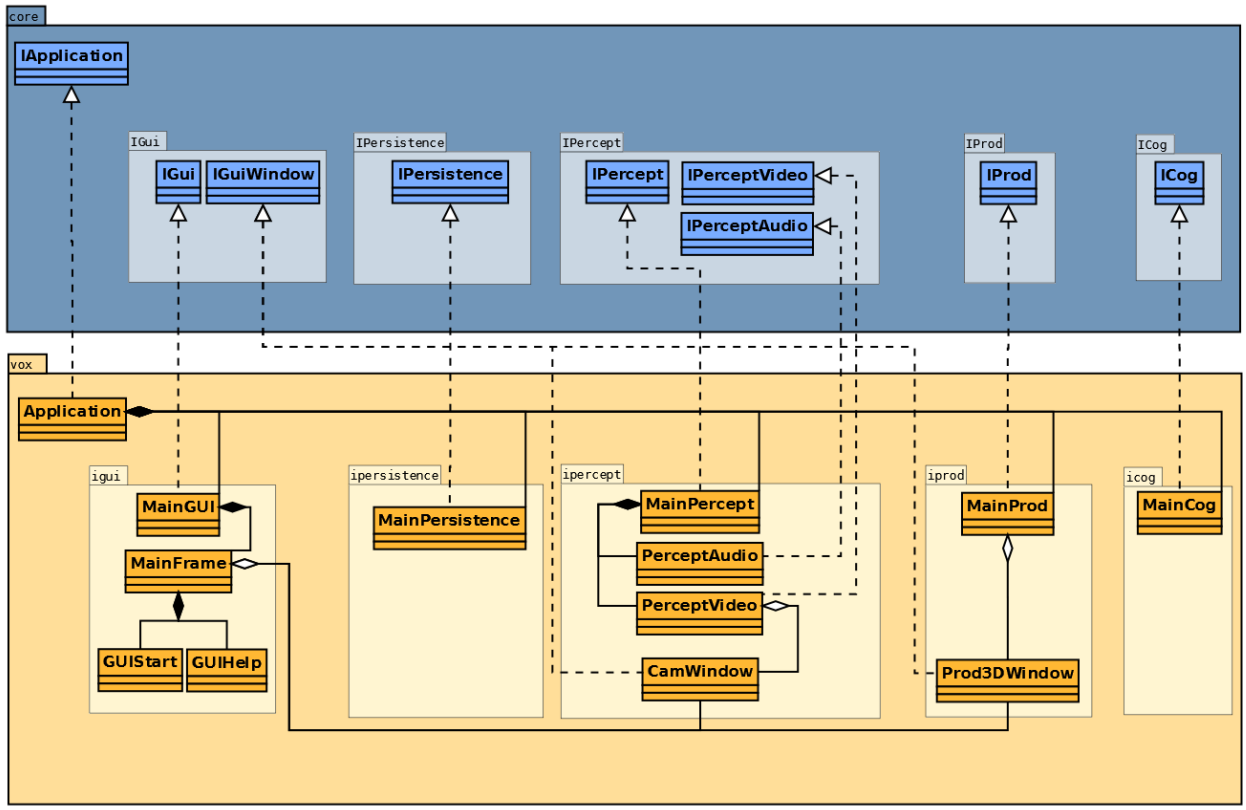


Figura 7.2: Diagrama de Clases UML. Resumen

En la figura 7.2 puede verse un esquema general en UML del proyecto completo. Es uns visión simplificada pero que refleja la arquitectura del sistema.

Se distinguen dos grandes bloques: En azul se encuentra el paquete core, que se compone de las interfaces separadas en módulos; en naranja la implementación de los distintos módulos y de la aplicación en sí. Cabe destacar que se persigue intencionadamente una alta modularidad y que en última instancia parte de estos módulos se ejecutarán en hilos independientes, aprovechando la tendencia actual de procesadores de multi-núcleo.

Debe tenerse en cuenta que los módulos implementados sólo podrán comunicarse entre ellos y con la aplicación a través de dicha interfaz. De esta forma se abstraen los detalles de la construcción de cada componente y se asegura su independencia, de forma que cualquier módulo pueda ser reemplazado o reimplementado de forma independiente sin que afecte al resto del proyecto. También permitiría la construcción de forma flexible de distintas aplicaciones, utlizando sólo los módulos necesarios o aprovechando módulos de otras ya creadas.

Diseño en UML - Núcleo de la Interfaz

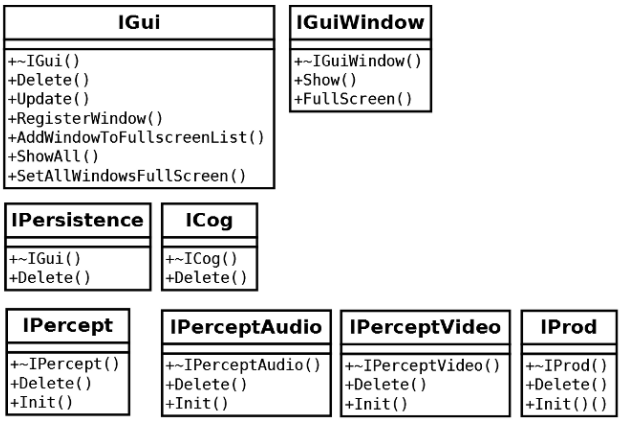


Figura 7.3: Diagrama de Clases UML. Detalle - core

La interfaz provee de unas herramientas básicas para la creación de cada núcleo y la comunicación entre ellos. Desde la aplicación y entre los módulos, toda interacción se hará a través de esta interfaz, aislando cada completamente dcada componente e los detalles de implementación de los otros.

Si bien en la fase actual no existe una gran funcionalidad, a medida que avance el proyecto se irán añadiendo nuevas capacidades que permitan, por ejemplo el paso de datos, la generación de contenido o la gestión de objetos.

En este apartado, el trabajo se concentra en los módulos de interfaz gráfica de usuario, percepción y producción. Con ellos se podrá disponer de un entorno con las capacidades básicas para la captura de lso datos necesarios y la visualización de los contenidos a generar.

Diseño en UML - Módulo de GUI

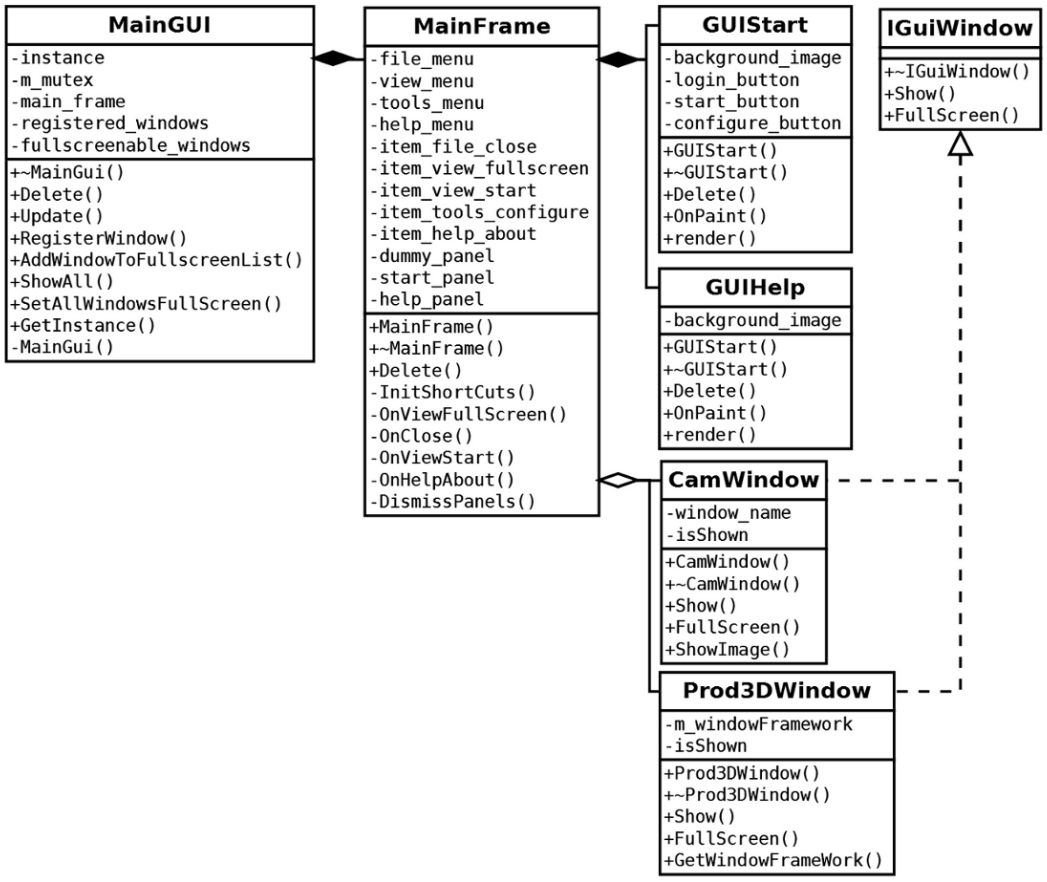


Figura 7.4: Diagrama de Clases UML. Detalle - igui

Este módulo controla gestión de la interfaz gráfica de usuario de tipo ventana de la aplicación.

La interfaz principal se define en la clase MainFrame, desde la que se gestionaría la aplicación con los menús y paneles principales. Por otro lado, hay que tener en cuenta que la aplicación está destinada a mostrar multitud de ventanas independientes, que podrían mostarse en distintas pantallas o proyectores. Un modelo MDI de ventanas flotantes libres del entorno principal es el más adecuado.

Hay que tener en cuenta que varias de las librerías a usar aportan mecanismos propios para la visualización de ventanas. Considerando también la simplicidad del uso de estos mecanismos y que las distintas ventanas visualizarían distintos tipos de datos, que pertencen a distintos módulos que se ejecutan en distintos hilos, se plantea por simplicidad definir en este módulo una clase IGUIWindow a modo de wrapper que englobe las funcionalidades que se requieren de los mismos, y que permita su gestión centralizada.

Diseño en UML - Módulo de Percepción

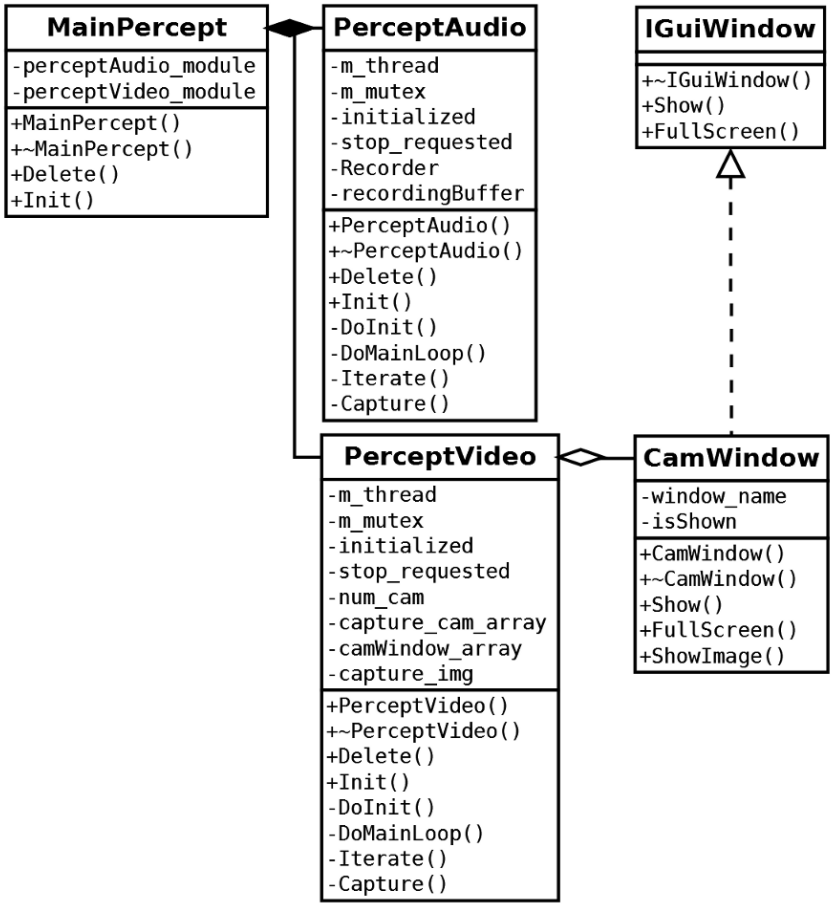


Figura 7.5: Diagrama de Clases UML. Detalle - ipercept

El módulo de percepción está destinado a la captura y análisis de los datos de entrada. Enel sistema que se plantea estos datos llegan a través de disntintas cámaras web y un micrófono. Por el momento, en esta primera demo sólo se procede a la captura de datos, dejando el análisis para fases posteriores.

Debido a la distinta naturaleza de entrada y procesado para video y audio, MainPercept lanza dos módulos que se ejecutan independientemente, cada uno en un hilo: PerceptAudio captura sonido y lo almacena en un buffer, y PerceptVideo captura imágenes. Este último está preparado para gestionar un número configurable de cámaras web y mostrar sus capturas en sus respectivas ventanas.

Hay que tener en cuenta que se crean las ventanas para visulizar las imágenes utilizando las herramientas de la propia librería. Esto se hace por simplicidad. Sin embargo, debe notarse que se utiliza el wrapper de ventanas IGuiWindow para encapsular las ventanas creadas con de esta forma y permitir su manipulación desde el módulo de interfaz gráfica de usuario.

Diseño en UML - Módulo de Producción

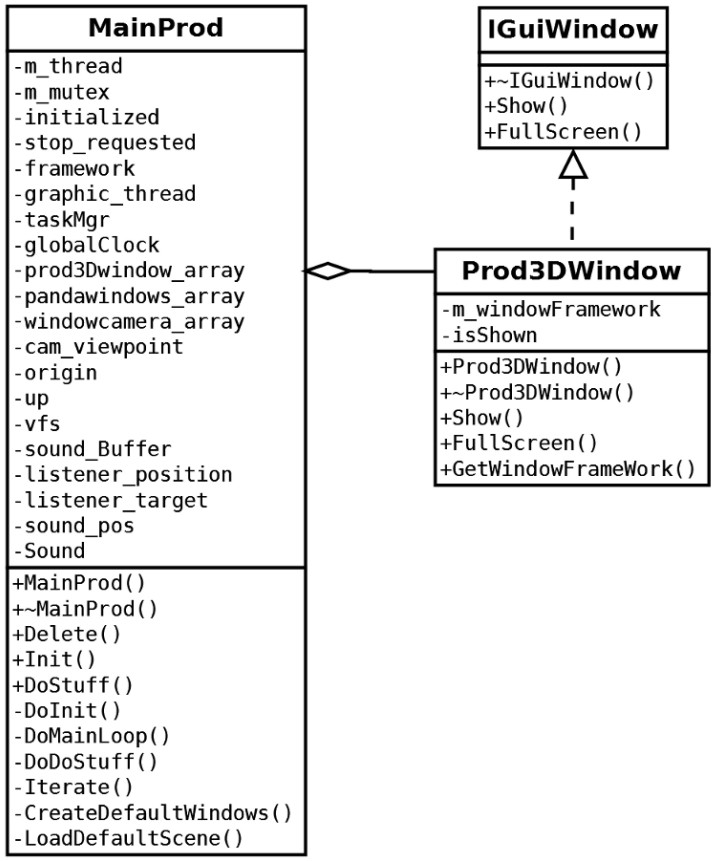


Figura 7.6: Diagrama de Clases UML. Detalle - iprod

MainProd es el módulo principal de Producción y también se ejecuta en un hilo independiente. Está destinado no sólo a la visualización del entorno sino a la generación de contenido, tanto gráficos como composición de audio. Sin embargo, en el alcance de esta fase el módulo sólo incorpora el primer apartado, la visualización y reproducción.

Aunque el módulo se ejecuta en un hilo independiente está preparado para que otros hilos intervengan mediante llamadas a unos métodos determinados. Además, de igual forma que lo hace el módulo de percepción, MainProd esta preparado para manejar un número configurable de ventanas, cuya intención es ser visualizadas en distintas pantallas o proyectores. Estas ventanas

también son creadas mediante las capacidades propias de la librería y utilizan IGuiWindow como envoltorio para su manipulación centralizada.

Por otro lado se definen en el espacio las fuentes de sonido y las propiedades del observador (listener) para disponder de sonido posicional en tres dimensiones.

7.3. Implementación

Se ha escogido como entorno de desarrollo Visual Studio 2008 Express Edition. Esto es debido a que se trata del entorno de desarrollo con que el se tiene mayor experiencia, así como por la comodidad de las herramientas de las que dispone. Se ha escogido la versión Express por ser gratuita y permitir el desarrollo de cualquier tipo de proyectos, tanto comerciales como no comerciales.

El proyecto consiste en una solución o colección de proyectos de VS. Estos proyectos son:

- core: Núcleo de la interfaz.
- icog: Proyecto para el Módulo Cognitivo.
- igui: Proyecto para el Módulo de GUI.
- ipercept: Proyecto para el Módulo de Percepción.
- ipersistence: Proyecto para el Módulo de Persistencia.
- iprod: Proyecto para el Módulo de Producción.
- vox: Aplicación.

El núcleo de la interfaz 'core' se implementa como una librería dinámica y es una dependencia necesaria para todos los proyectos. Por simplicidad los proyectos icog, igui, ipercept, ipersistence e iprod, que correspondería cada uno con un módulo, son librerías estáticas de las que depende la aplicación ejecutable 'vox'.

Hay que tener en cuenta que el IDE elegido no es multiplataforma, por lo que sólo puede ser utilizado bajo el sistema operativo Windows. Sin embargo, todas las librerías utilizadas sí lo son, y el código es C++, por lo que se debería poder crear nuevas soluciones para otros entornos sin grandes cambios. Se propone como alternativa para su futura portabilidad el IDE open source Eclipse o la configuración de una distribución preparada con CMAKE, de forma que se puedan generar proyectos para otros entornos. Sin embargo, esto quedará como trabajo futuro.

Finalmente, comentar que se decide seguir una estructura de ficheros que se corresponda directamente con la jerarquía de los proyectos. Además se generarán los resultados en un directorio común llamado \bin y las librerías externas a usar se almacenarán dentro del subdirectorio correspondiente dentro de la carpeta \extern. El código de los módulos implementados puede encontrarse

en \src, y el de la aplicación el \app.

Llegados a este punto se procede a revisar la implementación de cada apartado:

Núcleo de la Interfaz

\src\core

Contiene las cabeceras que definen clases virtuales puras, de las cuales heredarán las clases de los módulos que se implementen. Esto es así para asegurar la abstracción e independencia entre los detalles de implementación de cada módulo.

A continuación puede verse un ejemplo de su uso con la interfaz IPercept:

```
1 namespace core{
2 class IPercept
3 { public:
4    virtual ~IPercept(){}
5    virtual void Delete()=0;
6    virtual void Init()=0;
7 };}
```

Este ejemplo es válido para el resto de las interfaces, que incluyen en cada caso los métodos virtuales necesarios. Estos métodos implementados se corresponden directamente con los ilustrados en la figura 7.3, página 41.

Módulo de GUI

\src\igui

La clase principal del Módulo de GUI y que implementa la interfaz IGUI se llama MainGui. Esta clase es la que se encarga de toda la gestión de la interfaz de ventanas del proyecto y la librería escogida para su implementación es wxWidgets.

Las instrucciones para descargar y compilar wxWidgets pueden encontrarse en la web oficial [24]. Los paquetes descargados vienen preparados para VS2008 y compilan sin necesidad de ningún ajuste. Sólo hay que incluir las cabeceras y las librerías necesarias, que pueden verse en la sección B.4, página 114.

Hay que tener en cuenta que una de las particularidades de wxWidgets es que necesita tomar control de la aplicación principal, mediante la macro IMPLEMENT_APP. Este detalle puede verse en en profundidad en la sección 7.3, página 56. De todas formas, eso no afecta a la construcción y gestión de la interfaz, que se realizará completamente en este módulo.

Por otro lado, hay que tener en cuenta que distintos módulos tienen la capacidad de crear ventanas para mostrar información y que se desea usar esos mecanismos. Para ello se utiliza la interfaz IGuiWindow para encapsular la creación y la gestión, y se define MainGui como estática para poder acceder a la misma instancia desde cualquier módulo que lo necesite. Para asegurar que el acceso a la instancia sea seguro se utiliza un cerrojo como mecanismo de sincronización, implementado con boost.

- MainGui: Crea y mantiene los elementos de la interfaz. Se compone de un marco principal de aplicación llamado MainFrame y diversos paneles. Así mismo, también mantiene una lista de las ventanas independientes que se hayan registrado. Se crea la macro IMPLEMENT_MIIAPP para encapsular la macro de wxWidgets IMPLEMENT_APP. Un resumen de la clase puede verse en el siguiente cuadro:

```
#define IMPLEMENT_MIIAPP(name) IMPLEMENT_APP(name)
namespace core {namespace igui{
class MainGui : public core::IGui
{ public:    ...
  private:
    MainGui(const std::string &title = "");
    static MainGui* instance;
    static MainFrame *main_frame;
    static std::map<IGuiWindow*, int> registered_windows;
    static std::map<IGuiWindow*, int> fullscreenable_windows;
};}}

MainGui* MainGui::GetInstance(const std::string &title)
{ boost::mutex::scoped_lock lock(m_mutex);
  if (instance == NULL)
   instance = new MainGui(title);
  return instance; }
```

- MainFrame: Crea el marco principal de la aplicación. Contiene el menú y los paneles principales. En esta fase los únicos paneles que existen por el momento son el de Inicio y el de Ayuda. Se ha implementado un mecanismo para la sustitución de contenidos de ventanas, de forma que la ventana principal de la aplicación asumirá el contenido de los paneles que se usen en cada momento de forma dinámica.

```
namespace core { namespace igui {
class MainFrame : public wxFrame
{ public:    ...
 private:
    wxMenu *file_menu, *view_menu, *tools_menu, *help_menu;
    wxMenuItem *item_file_close, *item_view_fullscreen, *
       item_view_start, *item_tools_configure, *item_help_about;
```

```
7       ...
8       DECLARE_EVENT_TABLE()
9       void OnClose(wxCommandEvent& WXUNUSED(event));
10      void OnViewStart(wxCommandEvent& WXUNUSED(event));
11      void OnHelpAbout(wxCommandEvent& WXUNUSED(event));
12      void DismissPanels();};}}
```

- GUIStart: Es el panel principal de la aplicación que contendrá un acceso rápido a las funciones principales de la aplicación mediante los botones: Login, Inicio y Configuración.

  El aspecto más relevante de esta clase son los botones y la necesidad de capturar el evento de render EVT_PAINT para tener un acceso directo a cómo se dibuja el contenido de la misma. Esto es necesario para conseguir acabados más interesantes; por ejemplo, dibujar una imagen de fondo o texto con fondo transparente sobre imágenes. Debe tomarse el DC (contexto del dispositivo) con el que se va a dibujar, en este caso la pantalla, y usar las operaciones de dibujo directamente.

```
1  namespace core { namespace igui {
2  class GUIStart : public wxPanel
3  { public:   ...
4     void OnPaint(wxPaintEvent &evt);
5     void paintNow();
6     void render(wxDC& dc);
7   private:
8     wxBitmap background_image;
9     wxButton *login_button, *start_button, *configure_button;
10 };}}
11
12 BEGIN_EVENT_TABLE(GUIStart, wxPanel)
13    EVT_PAINT (GUIStart::OnPaint)
14 END_EVENT_TABLE()
15
16 GUIStart::GUIStart(...):wxPanel(parent, id, pos, size, style, name)
17 { background_image = wxBitmap(...);
18   login_button = new wxButton(...); }
19
20 void GUIStart::OnPaint(wxPaintEvent & evt)
21 { wxPaintDC dc(this);
22   render(dc); }
23
24 void GUIStart::render(wxDC& dc)
25 { dc.DrawBitmap(background_image, 0, -20, false ); }
```

- GUIHelp: Panel de información de la aplicación donde puede verse la versión y donde

añadirá un enlace a la web del proyecto. Los detalles de implementación son muy similares al panel GUIStart.

Para ver más detalles sobre la configuración de la librería, dependencias necesarias y la implementación del módulo se puede consultar la sección B.4, en la página 114.

Módulo de Percepción

\src\ipercept

La clase principal que implementa la interfaz IPercept y que está destinada a gestionar el módulo se llama MainPercept. Las tareas que se resuelven aquí son la captura de información del espacio de la instalación y de su interpretación. La entrada de datos del sistema consiste en audio mediante micrófonos, y de video, mediante webcams. En esta fase las tareas iniciales se centran en capturar audio y almacenarlo en un buffer, y capturar imágenes de las cámaras para mostarlas. Más adelante, se usará esta información para calibrar los puntos de vista o localizar al usuario dentro del espacio de la instalación para posicionarlo dentro del entorno virtual, entre otras funcionalidades.

Puede apreciarse que la captura de información y su procesado son de naturaleza distinta y, de hecho, independiente; no sólo del resto de la aplicación, sino entre sí. Además hay que tener en cuenta que los accesos a los periféricos son tareas costosas en tiempo. Por ello, la mejor opción es mantener estas tareas de forma independiente, ejecutándose en hilos separados. Cuando el sistema necesite alguna información, solicitará al módulo la más reciente, pero no será necesaria la espera para que éste termine de acceder a los dispositivos o de realizar sus tareas.

Las librerías utilizadas son SFML para la captura de audio, y OpenCV para la captura de video y su posterior proceso. Son librerías que han requerido ajustes importantes para su funcionamiento en el proyecto. Pueden verse los detalles en el anexo **??** página **??**.

- MainPercept: Implementa los mecanismos necesarios para la gestión global del módulo y de sus componentes, de tal forma que se pueda controlar la entrada de datos y su procesado. Por ahora sólo instancia los módulos necesarios y los inicializa.

```
1  namespace ipercept {
2  class MainPercept : public core::IPercept
3  { public:   ...
4    private:
5     static int num_cam;
6     static PerceptAudio* perceptAudio_module;
7     static PerceptVideo* perceptVideo_module;
8  };}}
9
10 MainPercept::MainPercept()
```

```
{ perceptAudio_module = new PerceptAudio();
  perceptVideo_module = new PerceptVideo(); }

void MainPercept::Init()
{ perceptAudio_module->Init();
  perceptVideo_module->Init(); }
```

- PerceptAudio: Esta clase implementa la interfaz IPerceptAudio y se encargará de ejecutar la captura y procesado de audio. Por ahora sólo captura y almacena el sonido en un buffer. Las librerías usadas son SFML para el acceso a los dispositivos de audio y grabación, y Boost para la gestión de hilos.

  Por coherencia se decide seguir un mismo esquema para los módulos de este tipo. Se tratan de clases con métodos y atributos estáticos que internamente lanzan un hilo independiente para la ejecución de su código principal. Todas tienen un método llamado Init(), que internamente llama a DoInit() para inicializar el módulo y ejecutar la llamada a DoMainloop(), el bucle principal, en un hilo aparte. Internamente Iterate() recoge el código destinado a ejecutarse para cada iteración, mientras que Capture() realiza la captura de datos. Se hace uso de cerrojos no bloqueantes definidos en contexto para el acceso a los atributos. Cuando la ejecución de hilo llega a este cerrojo se permite saltar el código bloqueante a la espera de que en la siguietne iteración el cerrojo esté disponible, por lo que el hilo no llega a bloquearse nunca.

```
class PerceptAudio : public core::IPerceptAudio
{ public:    ...
  private:
    static void DoInit();
    static void DoMainLoop();
    static void Iterate();
    static void Capture();

    static boost::shared_ptr<boost::thread> m_thread;
    static boost::try_mutex m_mutex;
    static bool initialized, stop_requested;

    static sf::SoundBufferRecorder Recorder;
    static sf::SoundBuffer recordingBuffer;};}}
```

- PerceptVideo: Esta clase implementa la interfaz IPerceptVideo y se encarga de ejecutar la captura y procesado de información visual. Por ahora la tarea consiste en capturar imágenes de un número configurable de cámaras web y mostrarlas en ventanas independientes. Como estás tareas son costosas en tiempo es altamente relevante que este módulo se ejecute en un hilo aparte. Las librerias a usar son Boost (B.2.2), para la gestión de hilos, y OpenCV (B.2.4),

librería de Visión por Computador, para la captura de imágenes y su posterior procesado.

En la construcción del módulo se sigue el mismo esquema que se ha visto en casos anteriores para mantener la coherencia. Así, se dispone de métodos para la inicialización y se separa la ejecución del bucle principal, que es lanzando en un hilo independiente, con cerrojos no bloqueantes. Así mismo, se distingue la ejecución de cada iteración, donde se incluye un método para la captura de datos de entrada. Se dispone de un vector de cámaras que permita la flexibildiad del módulo para recoger datos de un número configurable de fuentes.
A continuación se muestran algunos detalles relevantes.

```
class PerceptVideo : public core::IPerceptVideo
{ public:   ...
  private:
   static void DoInit();
   static void DoMainLoop();
   static void Iterate();
   static void Capture();

   static int num_cam;
   static std::map< int, CvCapture* > capture_cam_array;
   static std::map< std::string, CamWindow* > camWindow_array;};}}
```

A continuación se muestra un cuadro de código de resumen que ilustra el proceso de inicialización, creación del nuevo hilo de ejecución y el bucle principal del mismo, donde se capturan los datos de entrada.

```
PerceptVideo::PerceptVideo()
{ capture_cam_array[i] = cvCaptureFromCAM(i);
  camWindow_array[window_name] = new CamWindow(window_name);}}}

void PerceptVideo::DoInit()
{ assert(!m_thread);
  m_thread = boost::shared_ptr<boost::thread>(new boost::thread(
     boost::function0<void>(&PerceptVideo::DoMainLoop)));}}

void PerceptVideo::DoMainLoop()
{ while(!stop_requested)
  { Iterate();
    m_thread->sleep(system_time()+milliseconds(10));}}

void PerceptVideo::Iterate()
{  boost::try_mutex::scoped_try_lock lock(m_mutex);
   if (lock) {Capture();}}

```

```
18 void PerceptVideo::Capture()
19 { for (...)
20   { capture_img = cvQueryFrame(iter->second);
21     std::map<...>::iterator cam_iter = camWindow_array.find(
          window_name);
22     cam_iter->second->ShowImage(capture_img);}}
```

Para más información se puede consultar la sección B.5, página 117.

- CamWindow: Esta clase es utilizada para encapsular la creación de ventanas mediante las herramientas que provee OpenCV, de forma que se adapten a la interfaz IGUIWindows. Esto permitirá su acceso y gestión desde el módulo de GUI. Se elige usar los mecanismos que ofrece OpenCV para la creación de ventanas para simplificar el uso de las mismas y permitir su actualización a partir de las herramientas propias de la librería, sin necesidad de estar delegando ni transmitiendo información innecesariamente entre módulos.

```
1  namespace core { namespace ipercept {
2  class CamWindow : public core::IGuiWindow
3  { public:   ...
4    private:
5     std::string window_name;
6     bool isShown;
7  };}}
8
9  CamWindow::CamWindow(const std::string &_window_name)
10 { cvNamedWindow(window_name),1);
11   core::igui::MainGui::GetInstance()->RegisterWindow(((core::
        IGuiWindow*)this)); }
12
13 void CamWindow::ShowImage(const IplImage *image)
14 { if (isShown) cvShowImage(window_name, image);}
```

Módulo de Producción

\src\iprod

Siguiendo el mismo modelo que en los módulos comentados anteriormente, la clase principal que implementa la interfaz IProd recibe el nombre de MainProd. Es a través de esta interfaz como se manejará el contenido a generar para construir y exponer el entorno. En esta primera demo el obejtivo es integrar un motor gráfico que sea capaz de visualizar una escena 3D por defecto, con sonido 3D en un sistema envolvente, mostrando una cantidad configurable de ventanas.

De la misma forma que en el resto de los casos, este módulo se ejecuta en un hilo independiente, utilizando los mecanismos de sincronización necesarios para el manejo de hilos gracias a

la librería Boost (B.2.2). Por otro lado, el motor gráfico integrado es Panda3D. Sin embargo, es necesario destacar que fueron necesarios ajustes en las opciones de compilación del mismo para que las librerias generadas pudieran funcionar correctamente dentro del proyecto. También impone algunas limitaciones como una política restrictiva en el nombrado y jerarquía de directorios, así como algunas modificaciones en los fuentes poco comunes pero que fueron necesarias realizar y aceptadas en la revisión oficial. Para más detalles, consultar B.2.5, página 111.

Finalmente para la localización de las fuentes de sonido y del oyente en el espacio 3D, se ha hecho uso de la librería SFML. De nuevo, es necesario destacar la necesidad de realizar modificaciones en el código fuente de la libreria para que ésta funcione correctamente. Esto es debido que existen errores en el diseño de la API que impide que pueda situarse libremente al oyente en el espacio 3D. Para ver más detalles sobre la integración de esta librería y las modificaciones necesarias se puede consultar la sección B.2.3, página 108.

- MainProd: Implementa la interfaz IProd y provee los mecanismos para cargar una escena 3D y mostrarla en una cantidad de ventanas configurable. También carga y localiza en el espacio sonidos y el oyente.

```
1  namespace core { namespace iprod {
2  class MainProd : public core::IProd
3  { public: ...
4    private:
5     static void CreateDefaultWindows(int numWindows);
6     static void LoadDefaultScene();
7
8     static PandaFramework framework;
9     static std::map<int, WindowFramework*> pandawindows_array;
10    static std::map<int, NodePath>         windowcamera_array;
11    static NodePath cam_viewpoint, origin, up;
12    static double listener_position[],listener_target[],sound_pos[];
13    static sf::Sound Sound;
14 };}}
```

Las secciones más relevantes son la carga de la escena y el render de las distintas vistas. Por otro lado, se tienen en cuenta políticas de rendimiento y se trata que la escena sea ligera. Por ejemplo, los objetos se cargan una sóla vez para el marco principal y se instancian como referencias en los grafos de escena del resto de vistas, sin necesidad de cargar varias veces el mismo modelo.

Para poder realizar pruebas para comprobar el buen funcionamiento y para preparar trabajo futuro, se añade un mecanismo para el acceso externo a la escena, donde realizar operaciones DoStuff(). En este caso, rotar la cámara. Así mismo, se habilita la navegación libre en la segunda ventana de visualización mediante el uso del ratón (para comprobar que el correcto

funcionamiento de la localización 3D tanto de oyente como de sonidos posicionados).

Gran parte del código se asemeja a los módulos explicados anteriormente, siguiendo por coherencia la misma política de incialización, ejecución del hilo independiente, y bloques de codigo a ejecutar en cada iteración. por lo que sólo se detallanA continuación se muestran las secciones de código más relevantes:

```
void MainProd::DoMainLoop()
{ framework.open_framework(m_argc,m_argv);
  CreateDefaultWindows(DEFAULT_NUM_WINDOWS);
  LoadDefaultScene();
  sf::SoundBuffer Buffer;
  Sound.SetBuffer(Buffer);

  while(!stop_requested)
  { Iterate();
    m_thread->sleep(get_system_time()+milliseconds(10)); }

  framework.close_all_windows();
  framework.close_framework();}
```

Algo a destacar es la función Iterate(). Por lo general, los motores gráficos y de juego toman posesión del bucle principal de la aplicación. Este también es el caso del motor de juego elegido, Panda3D. Esta peculiaridad limita el número de actividad y la facilidad con la que nuevas tareas pueden ser añadidas, o no sólo añadidas, sino generadas mientras se ejecuta la aplicación. Sin embargo, la flexibilidad del motor escogido permite abstraerse de las secciones de inicialización y del bucle principal, de forma que se pueden ejecutar las tareas de cada iteración mediante el método step(). Gracias a este mecanismo podemos disponer de la ejecución del bucle principal en un hilo aparte, mientras se ejecutan o generan tareas y objetos dinámicamente y de forma sincronizada, desde el mismo u otros hilos.

```
void MainProd::Iterate()
{ boost::try_mutex::scoped_try_lock lock(m_mutex);
  if (lock)
  { framework.do_frame(graphic_thread);
    CIntervalManager::get_global_ptr()->step();}}

void MainProd::DoInit()
{ if (!initialized)
        { assert(!m_thread);
      m_thread = boost::shared_ptr<boost::thread>(new boost::thread(
         boost::function0<void>(&MainProd::DoMainLoop)));}}
```

Como se comentaba, el siguiente fragmento de código muestra la forma en que, desde la

ejecución de otro hilo puede solicitarse a la clase realizar determinadas tareas. En este caso, se rota la cámara de la primera vista para que gire entorno al centro de la escena, a la vez que se actualiza la posición del oyente en el espacio 3D según la posición de la cámara, con navegación mediante el movimiento del ratón, de la vista2.

Es necesario tener en cuenta dos puntos en relación al sonido 3D en la aplicación para un sistema envolvente. Por un lado los sistemas de referencia espacial del motor gráfico y del motor de audio no coinciden, por lo que es necesario realizar las transformaciones pertinentes. Por otro lado, la librería de audio utilizada comete errores importantes que deben ser solucionados para poder disponer de un posicionamiento y orientación correctos de las fuentes de sonido y del oyente. Estos detallen pueden verse con mayor profundidad en la sección B.2.3, página 108.

```
1  void MainProd::DoDoStuff()
2  { boost::try_mutex::scoped_try_lock lock(m_mutex);
3    if (lock && initialized)
4    { //rotar la cámara en vista 1
5      windowcamera_array[1].set_pos(20*sin(angleradians),-20.0*cos(
           angleradians),3);
6      windowcamera_array[1].set_hpr(angledegrees, 0, 0);
7      //actualizar oyente según navegación en vista 2
8      sf::Listener::SetPosition(new_pos.get_x(), new_pos.get_y(),
           new_pos.get_z());
9      sf::Listener::SetTarget(new_at.get_x(), new_at.get_y(), new_at.
           get_z() , new_up.get_x(), new_up.get_y(), new_up.get_z());
10    }
11    else
12    { //no se pudo coger el cerrojo, pero no se bloquea  }
13 }
```

Finalmente se muestra fragmentos que ilustran la carga de la escena y cómo se enlazan a los disintos renderers. Como se comentaba inicialmente, y como es lógico, se ha tenido en cuenta cuestiones de eficiencia para visualización en múltiples vistas. Como en el problema al que nos enfrentamos la escena a visualizar es la misma para todas las vistas, se sigue la política de cargar una única vez los elementos en la escena, ligarlos al render principal e instanciarlos para el resto, de forma que sólo se cargan una vez y no existen copias del mismo objeto en los distintos grafos de la escena.

```
1  void MainProd::LoadDefaultScene()
2  { if (pandawindows_array.begin() != pandawindows_array.end())
3    { NodePath environment = pandawindows_array[1]->load_model(
         framework.get_models(),"environment");
4      environment.reparent_to(pandawindows_array[1]->get_render());
5      NodePath pandaActor = pandawindows_array[1]->load_model(
```

```
        framework.get_models(), "panda-model");
    pandaActor.reparent_to(pandawindows_array[1]->get_render());
    pandawindows_array[1]->load_model(pandaActor, "panda-walk4");
    pandawindows_array[1]->loop_animations(0);

    std::map<...>::iterator iter = pandawindows_array.begin(); iter
        ++;
    while(iter != pandawindows_array.end())
    { environment.instance_to(iter->second->get_render());
      iter->second->setup_trackball();
      iter++;}}}
```

- Prod3DWindow: Esta clase es utilizada para encapsular la creación de ventanas mediante las herramientas que provee Panda3D, de forma que se adapten a la interfaz IGUIWindows. Esto permitiría su acceso y gestión desde el módulo de GUI. De la misma forma que sucede en el módulo de Percepción para la visualización de la entrada de datos en distintas ventanas, se utiliza por simplicidad las herramientas propias de la librería para la creación de las mismas y la actualización de su contenido.

```
namespace core { namespace iprod {
class Prod3DWindow : public core::IGuiWindow
{ public:
   WindowFramework *GetWindowFrameWork() {return m_windowFramework;}
  private:
   bool isShown;
   WindowFramework *m_windowFramework;};}}
```

Aplicación Principal

\apps\vox

Finalmente se muestra la sección correspondiente a la aplicación principal. Su objetivo es enmarcar los múltiples módulos, gestionar la ejecución de la misma y de sus componentes.

Algo a tener en cuenta es que al usar wxWidgets como librería para crear la interfaz de usuario de la aplicación, se impone un requisito incómodo: wxwidgets necesita tomar control de la aplicación principal. Esto impide la completa independencia entre la aplicación y la librería usada para la GUI. Sin embargo, hay que tener en cuenta que la Aplicación principal no hará nada: esta sección está únicamente destinada la carga de los distinto módulos. Por ello, su complejidad y contenido son mínimos. Es por ello que se decide proseguir.
En el caso de desear cambiar el módulo de GUI, son dos los cambios necesarios en la aplicación: eliminar la herencia de la clase wxApp, y sustituir la macro IMPLEMENT_MIIAPP(Application)

por un cuerpo main() donde se cree una instancia de la clase Application.

A continuación se muestran algunos detalles relevantes:

- Application: Implementa la interfaz IApplication que se usa para la creación de la aplicación. Su cometido es cargar los distintos módulos y proveer herramientas para su gestión. En caso de no desear usar wxWidgets como librería para la GUI, es necesario eliminar la herencia de la clase a wxApp.

  También hay que tener en cuenta que, en Windows, por motivos de incompitibilidades entre librerias, es necesario incluir la cabecera winsock2.h al inicio del fichero. Para más detalles se puede consultar la sección B.1, página 107

```
#ifdef _WINDOWS
#include <winsock2.h>
#endif
class Application : public wxApp, public core::IApplication
{ public:    ...
  private:
   core::IGui           *app_maingui;
   core::IPercept       *app_mainpercept;
   core::IProd          *app_mainprod;};

bool Application::OnInit()
{ app_maingui = MainGui::GetInstance("VOX");
  app_mainpercept=(core::IPercept*)new core::ipercept::MainPercept()
     ;
  app_mainprod=(core::IProd*)new core::iprod::MainProd(argc, argv);
  app_mainpercept->Init();
  app_mainprod->Init();
  return true; }
```

- Main: Fuente en el que se define el punto de entrada de la aplicación. En caso de no desear usar wxWidgets es necesario sustituir la línea IMPLEMENT_MIIAPP(Application) por la definición de una función main(), donde crear una instancia de la clase Application.

```
#define _WINSOCKAPI_
#include "Application.h"
IMPLEMENT_MIIAPP(Application)
```

7.4. Validación y Publicidad

7.4.1. Validación

Se sigue una filosofia de pruebas continuas para resolver en el momento en que aparecen los incidentes que puedan surgir al hacer cambios. Así mismo, cuando se añade una nueva funcionalidad, se usan mecanismos para comprobar que que las capacidades incorporadas funcionan correctamente y se mantiene un buen rendimiento.

- Comprobación de uso de recursos de la máquina mediante las herramientas del sistema. En la máquina en la que se desarrolla la aplicación muestra consumir un 5 % de CPU y 75Mb de memoria mantenidos. Hay que destacar la posibilidiad de que la aplicación consuma mayor CPU en máquinas con menor cantidad de procesadores debido al overhead introducido por la gestión de hilos. Sin embargo, la tendencia actual se mueve hacia el aumento de número de procesadores.

- Comprobración de la ejecución y cierre correctos, sin salidas de la aplicación inesperadas ni memory leaks.

- Uso de herramientas para medición de frames por segundo para comprobar el rendimiento de la ventana de render. Se comprueba que se mantiene siempre a 60fps para dos ventanas, y 30fps con hasta 10 ventanas de render. Hay que tener en cuenta que 60 es el límite máximo impuesto por la sincronización vertical del monitor.

- Uso de distintas opciones de configuración para comprobar las variantes del uso de los módulos, por ahora mediante macros: Abrir distinto número de ventanas de render (entre 1 y 10) y captura desde distintas cantidades de cámaras web (entre 1 y 2).

Las características de la máquina de referencia son las siguientes:

- Procesador Intel Core i7 CPU 870 2.92Ghz (4 núcleos reales con hyperthreading, símil 8 núcleos).

- 6Gb de Memoria Principal.

- Sistema Operativo Windows 7 64 bits

- Tarjeta gráfica GeForce GTS 240

Finalmente, se muestran capturas de ejemplo del estado actual de la aplicación, como pueden verse en las siguientes imágenes:

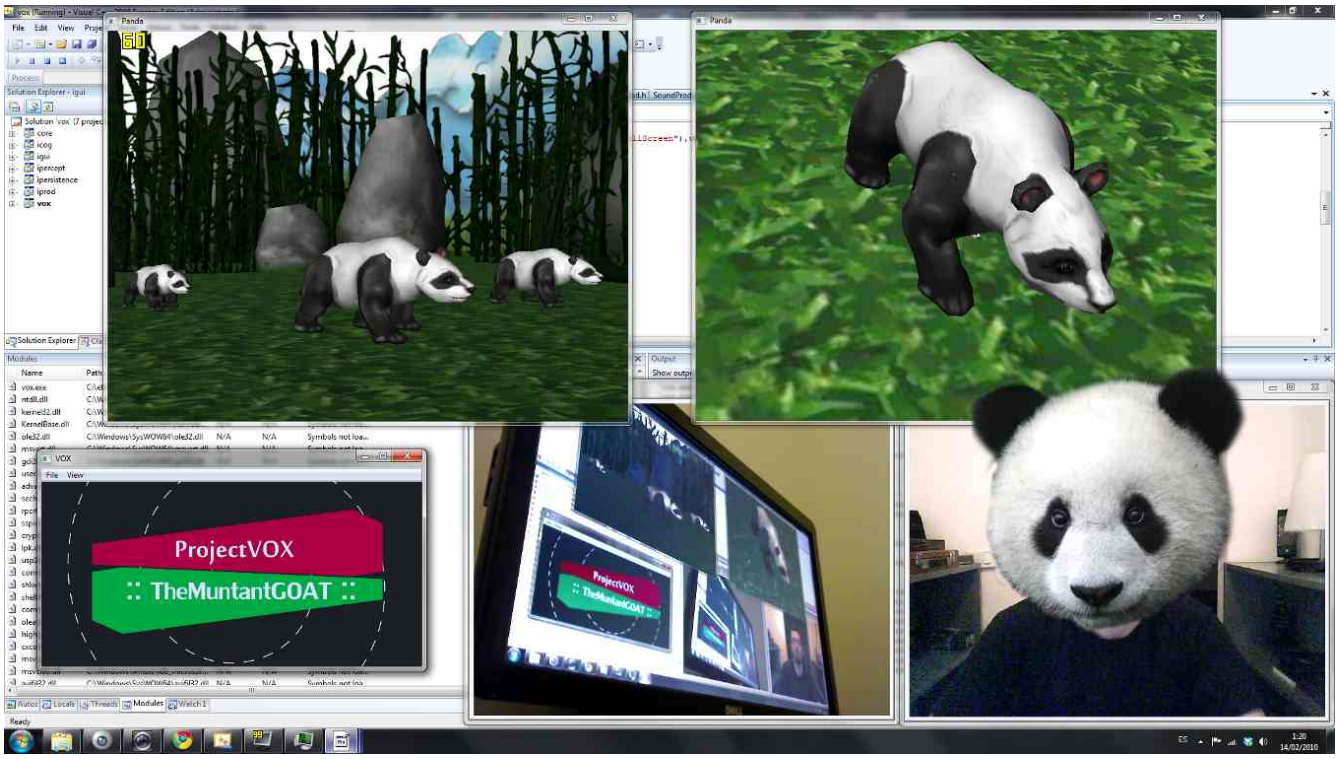


Figura 7.7: Captura 1 - Fase2: Primera Demo

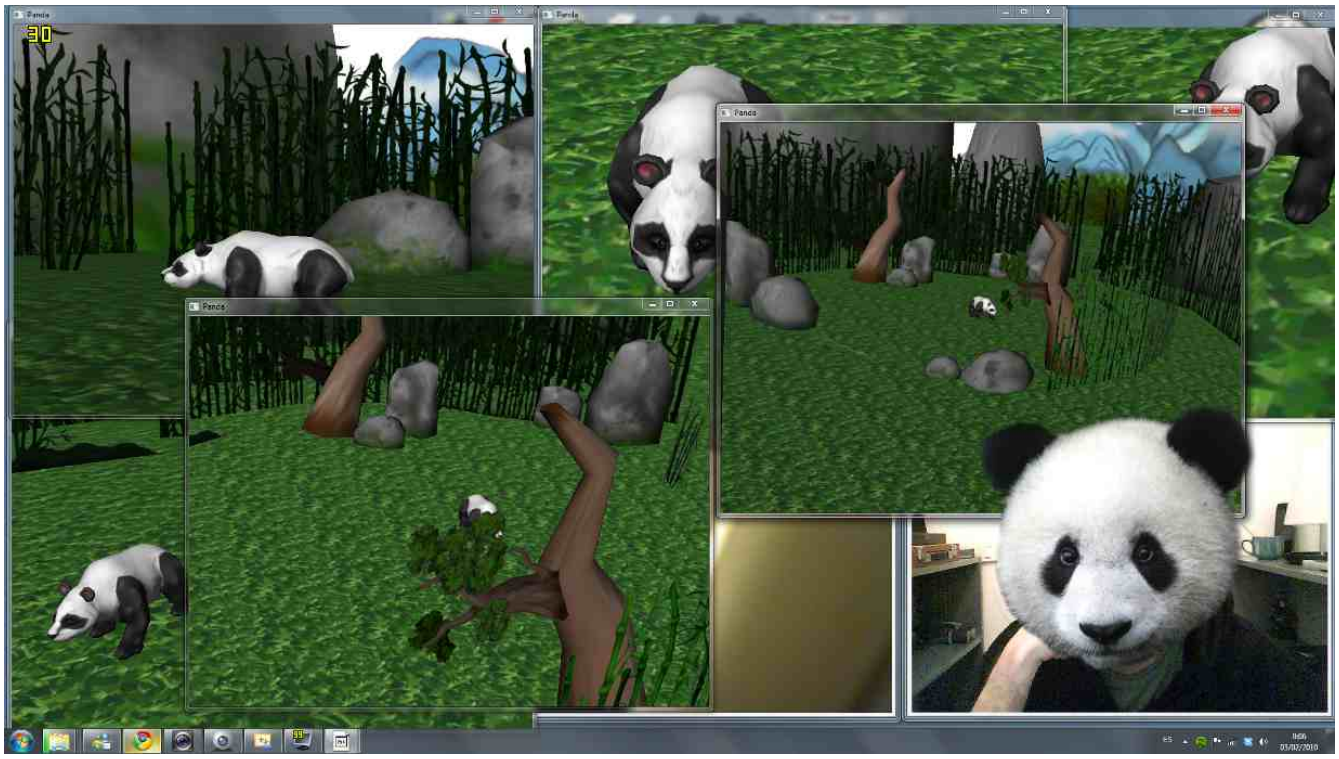


Figura 7.8: Captura 2 - Fase2: Primera Demo

7.4.2. Publicidad

Aunque el desarrollo del proyecto se encuentra aún en sus inicios se considera interesante tener una visión del panorama actual y valorar distintas opciones de publicidad e incluso posibles destinos para la exposición de la instalación. Además, se considera la creación de un portal web para la iniciativa, donde alojar el el entorno de trabajo resultante, asi como el proyecto.

Se valora establecer contacto con las siguientes instituciones:

- ULPGC.
- CAAM: Centro Atlántico de Arte Moderno.
- Gran Canaria Espacio Digital.
- Museo Elder de la Ciencia y la Tecnología.
- La Casa Encendida.

Posibles festivales:

- Artfutura

- Arco
- Estampa
- Sónar

Capítulo 8

Etapa 3: Usuario, Entorno, Configuración y Persistencia

Partiendo de la primera demo estable, se desea añadir añadir nuevas funcionalidades para completar la aplicación. En este punto se dispone de la estructura básica de la aplicación y de las herramientas necesarias para la captura de información y la reproducción de las vistas. Ahora se abordará el problema de la creación de usuarios, proyectos, escenas y su gestión. También se trabajará en las opciones de configuración del sistema básicas.

El punto más relevante en esta etapa consiste en el modelo de persistencia a seguir. Para ello, se propone hacer uso de una base de datos para guardar a los usuarios y sus escenarios, teniendo en mente la posibilidad de implementar en un futuro un sistema multiusuario persistente. Para este tipo de aplicación, convendría conseguir una Base de Datos Orientada a Objetos, a ser posible en Tiempo Real o con un buen tiempo de respuesta.

Entre otras ventajas, las OODBMS permiten eliminar la necesidad de un doble modelado de la información. El modelo de la base de datos se corresponde directamente con el modelo de datos de la aplicación. De esta forma se ahorra tiempo de diseño y se evita el problema del desajuste por impedancia. Además no es necesaria ninguna transformación explícita del modelo de datos. Por otro lado, se observan mejores respuestas en estos tipos de bases de datos, especialmente en el caso de accesos de tipo navigacional, es decir, aquellos en los que se accede a objetos a partir de relaciones con otros objetos, caso de esta aplicación.

Sin embargo, tras el estudio realizado no ha sido posible encontrar una herramienta que encaje con los requisitos software del proyecto. De entre las candidatas, EyeDB destacaba por ser la más prometedora; sin embargo, actualmente sólo está disponible para sistemas Linux. Del resto, o no disponían de licencias gratuitas, éstas eran muy limitadas, no disponían de APIs para C++ o las librerías eran antiguas y estaban abandonadas. Finalmente, se decidió apostar por usar una opción intermedia.

Finalmente, se usará una BBDD relacional (PostgreSQL), con un mapper objeto-relacional

(Debea). Aunque no se disponga de las ventajas de tener una base de datos directamente implementada como Orientada a Objetos, Postgre es actualmente consideraba la mejor opción libre para Bases de Datos Relacionales: potente, veloz y con un buen soporte de transacciones. Por otro lado, con el mapper objeto-relacional Debea se conservarán los beneficios para el desarrollo al eliminar la traducción del modelo OO al modelo relacional y simplificar la capa de persistencia. De esta forma se relega el problema del desajuste por impedancia al mapper. Con esto se espera reducir significativamente el tiempo de desarrollo y mantener un modelo fácil de mantener y flexible frente a cambios futuros.

8.1. Análisis

8.1.1. Análisis de Requisitos de Usuario

El subconjunto de funcionalidades añadidas se centran en la gestión de usuarios y de escenas con sus Entidades, además del equema de persistencia y la configuración de la aplicación.

En el apartado de Gestión, la aplicación permitirá dar de alta nuevos usuarios, crear o cargar entornos con entidades asociadas y editar sus propiedades.

De esta forma se permite la creación y borrado de usuarios, la modificación de permisos y que éste haga login en la aplicación. Un usuario registrado puede crear nuevos escenarios, modificar sus permisos, cargarlos o eliminarlos del sistema. Finalmente, esta información será guardada y cargada en futuras sesiones siguiendo el esquema de persistencia discutido. Sin embargo, las escenas y elementos serán objetos de prueba generados manualmente. Tampoco existirá interacción en esta fase.

En relación a la configuración de la aplicación se permitirá guardar datos relacionados a la configuración del sistema instalado en el equipo. Se utilizará para obtener directorios de uso común por los distintos módulos. También se usará para guardar y recuperar la configuración de las cámaras (número, orientación), así como de las ventanas de render (número, resolución, orientación). Se incluye de la misma forma una ventana de log donde los distintos módulos puedan volcar información útil.

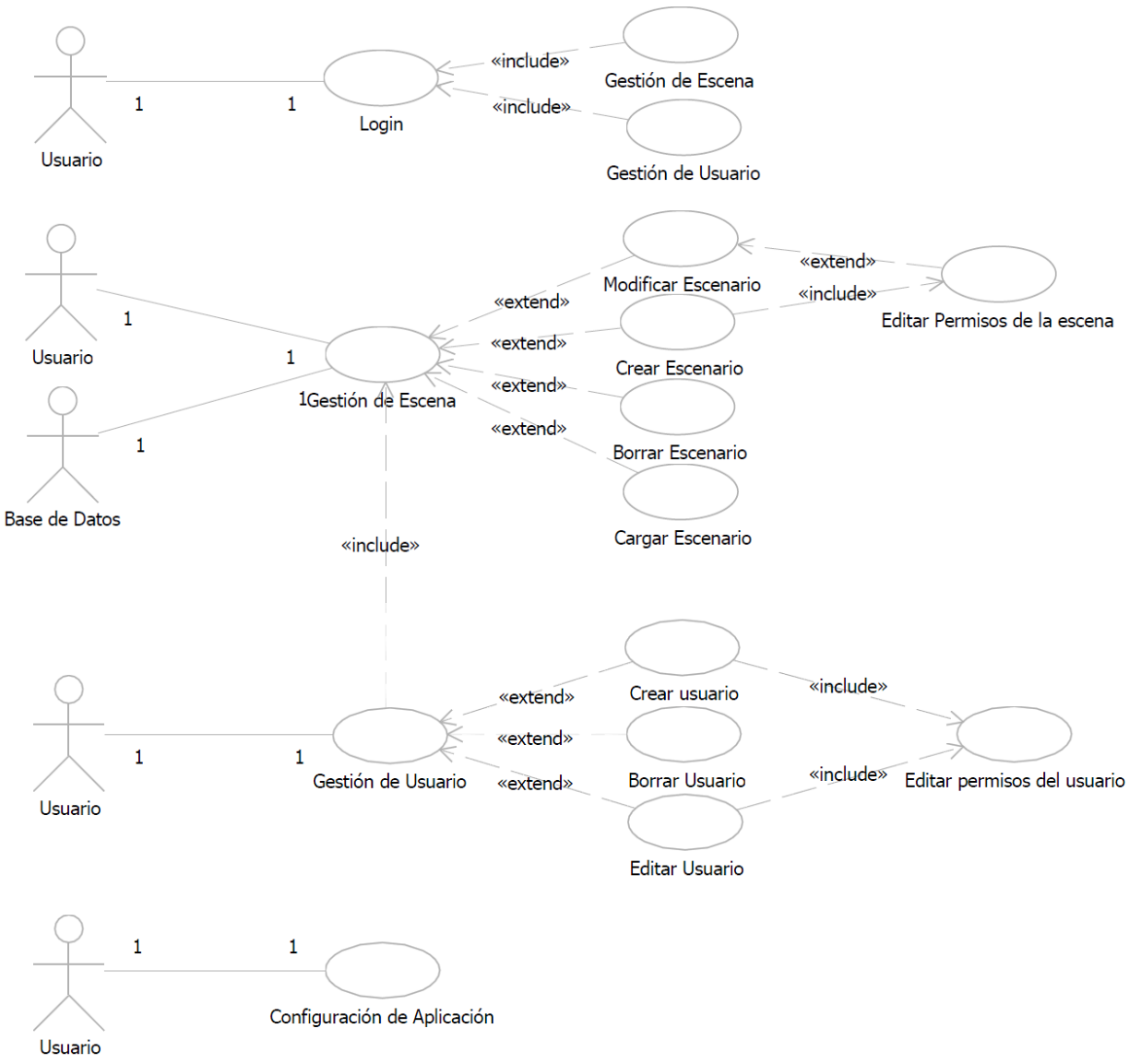


Figura 8.1: Diagrama de Casos de Uso. Usuario, Entorno, Configuración y Persistencia

8.1.2. Selección de Herramientas

A partir del estudio de las herramientas disponibles se han seleccionado las más adecuadas, tanto por sus características como por compatibilidades técnicas en el conjunto del proyecto.

- BBDD: PostgreSQL
- Mapper O-R: Debea

8.2. Diseño

8.2.1. Diseño de la Aplicación

Se añaden elementos al módulo IAplication para manejar los conceptos de ususario y proyecto, y se describe el módulo IPersistence.

- Core: Conjunto de interfaces de la aplicación.
  - IAplicationConfiguration: Se añade interfaz para la configuración de la aplicación.
  - IPersistence: Interfaz de la capa de Persistencia para el guardado y recuperación de los datos de la aplicación.
    - IUserPersistence: Interfaz que refleja del modelo de datos y persistencia del objeto Usuario.
    - IWorldPersistence: Interfaz que refleja del modelo de datos y persistencia del objeto Mundo (escenario).
    - IEntityPersistence: Interfaz que refleja del modelo de datos y persistencia del objeto Entidad. Una entidad es una elemento contenido en un escenario.
- igui: Interfaz básica para la creación del módulo y registro de ventanas. Modificaciones para soporte de opciones de configuración. Login, inicio y cierre de sesión de usuarios.
  - GUIConfiguration: Paneles de Configuración de la aplicación.
  - GUILogPanel: Panel de log donde se muestra información de la aplicación de los módulos que vuelquen datos en él.
  - GUIUser: Panel de login y de creación de nuevos usuarios.
  - GUIUserInfo: Panel de gestión de usuarios y creación y gestión de escenarios.
- ipersistence: Módulo de persistencia
  - EntityPersistence: Refleja el modelo de datos y la persistencia del objeto Entidad.
  - WorldPersistence: Refleja el modelo de datos y la persistencia del objeto Mundo.
  - UserPersistence: Refleja el modelo de datos y la persistencia del objeto Usuario.
- iprod: Módulo de producción
  - Prod3DEntity: Encapsula el modelo de datos del objeto Entidad y gestiona las características particulares relativas al módulo de producción.
- vox:
  - ApplicationConfiguration: Implementación de los mecanismos para guardar y cargar las opciones de configuración de la aplicación.

Breve descripción de los módulos

Los cambios principales se realizan en los módulos de Aplicación, GUI y Persistencia.

En el primero se introducen los controladores de sesión y de configuración y la API necesaria para su uso desde otros módulos. En GUI, se añade un controlador genérico y las interfaces gráficas necesarias para la configuración del sistema y para el inicio de sesión en la misma por parte de usuario. Finalmente, en el módulo de Persistencia se modelan los objetos a conservar y los mecanismos para poder guardarlos y recuperarlos en la base de datos.

Por otro lado, tanto el módulo de persistencia como el de GUI, así como el de percepción y producción reciben ajustes para poder usar las opciones de configuración.

Como modelo, recordar que cada módulo es responsable de las tareas a su cargo y las comunicaciones se establecen a través de los módulos principales. Ellos disponen de todos los mecanismos necesarios para realizarlas, delegando en su caso en controladores, y ofreciendo una API sencilla de usar. I.e., cuando el usuario introduce los datos de inicio de sesión el controlador de GUI recibe los datos desde el panel de la interfaz, que termina accediendo al módulo principal de GUI para que llame al módulo principal de aplicación, que es el responsable de gestionar la sesión. En este punto, la aplicación hace uso del controlador de sesión que terminará accediendo al módulo de persistencia para recuperar los datos.

Diseño General en UML

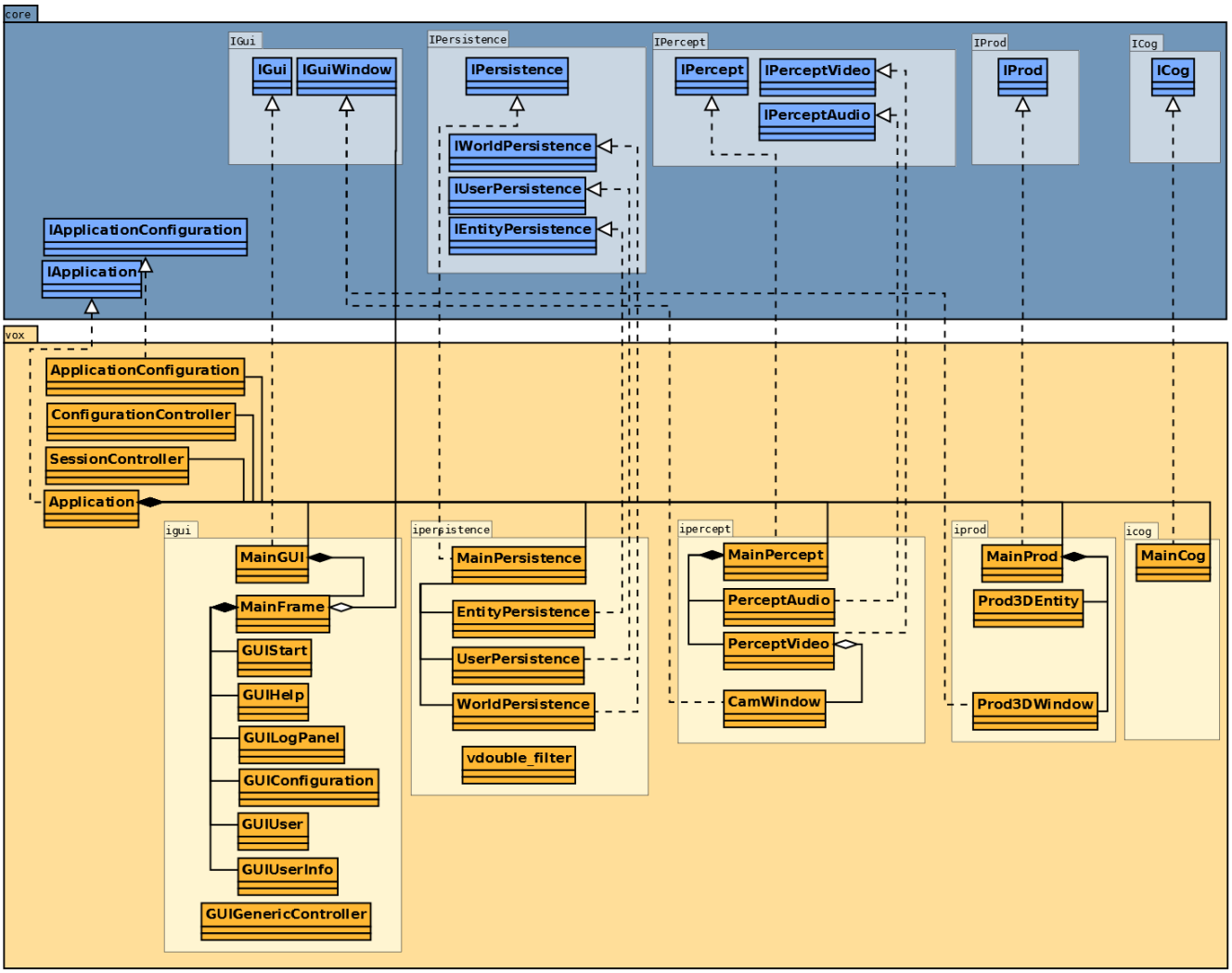


Figura 8.2: Diagrama de Clases UML, Segunda Demo. Resumen.

En este momento se aborda de lleno el módulo de persistencia, añadiendo los elementos Usuario, Mundo (Escenario) y Entidad. Este módulo será accedido para cargar o guardar los datos que se desean conservar. Por otro lado, también se da la posibilidad de configurar elementos de la aplicación, como pueden ser características relacionadas a los dispositivos de entrada o salida del sistema. Además se añade la capacidad de hacer login en la aplicación para cargar los mundos que tenga asociados un usuario, crear escenarios nuevos o editar propiedades del mismo.

Por simplicidad, en el diagrama se muestran sólo los detalles principales. Para ver los detalles se puede consultar las secciones siguientes.

Diseño en UML - Núcleo de la Interfaz

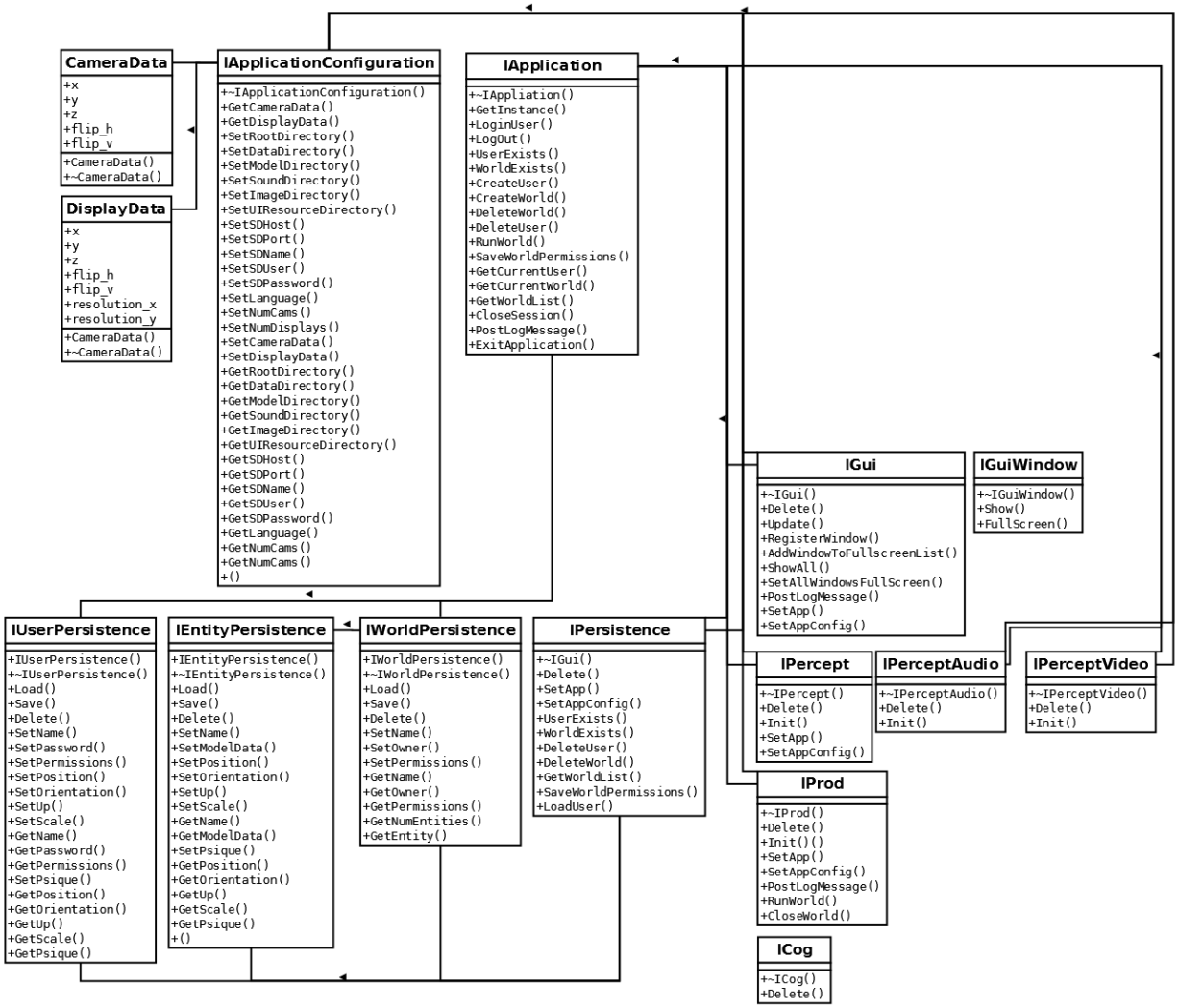


Figura 8.3: Diagrama de Clases UML, Segunda Demo. Núcleo de la Interfaz.

Se introducen cambios de relevancia en la mayoría de las clases, especialmente debido al uso de la interfaz de configuración que usarán los módulos. También cabe destacar el desarrollo de la interfaz de Persistencia que usarán aquéllos módulos que requieran acceder o guardar datos que se deseen conservar.

IApplicationConfiguration está destinada a sevir de interfaz para conservar datos de directorios de relevancia, como pueden ser el de datos genéricos, recursos de la aplicación, modelos 3D, sonidos o imágenes, lenguaje, datos de conexión con el dispositivo de almacenamiento, el número de cámaras, el número de displays, además de la configuración particular de los mismos (localización

en el espacio de la instalación, orientación y resolución).

Por otro lado tenemos las interfaces de persistencia. Es interesante recordar que se tiene como objetivo independizar los detalles de implementación de los módulos entre sí, de forma que sean fácilmente sustituíbles por otros nuevos, sin necesidad de realizar modificaciones en los demás. También se ha optado por un diseño de persistencia en el que existe una correspondencia directa entre el modelo OO y el modelo de datos a persistir. Por ello el modelo fundamental de los datos recae en el mismo módulo de persistencia. Como puede verse, es un esquema bastante simple en el que existen tres objetos: usuarios, mundos y entidades.

Los usuarios tienen los datos clásicos de conexión, para hacer login en el sistema pero también datos relativos a su experiencia dentro del mismo, como pueden ser desde la localización o permisos, hasta el término nombrado como psique. Psique es una codificación de la conducta y actividades que realiza el usuario en el sistema que será abordada más adelante. Para más detalles al respecto, consultar sección ?? página ??.

Las entidades son los elementos componentes de un mundo. Hasta este punto sólo se ha tenido en cuenta que sean elementos 3D posicionados en el espacio. Estas entidades también actúan y sufren experiencias que se conservará en la codificación psique, que a su vez influirá en sus acciones siguientes.

Finalmente, los mundos son espacios creados por un usuario y que contendrán colecciones de entidades conservando una visión global de la experiencia en su atributo psique. Como es razonable desear que un escenario permanezca inalterable, o poder acceder sólo a determinadas opciones en él, el usuario puede definir unos permisos sobre cada mundo.

Diseño en UML - Módulo de Aplicación

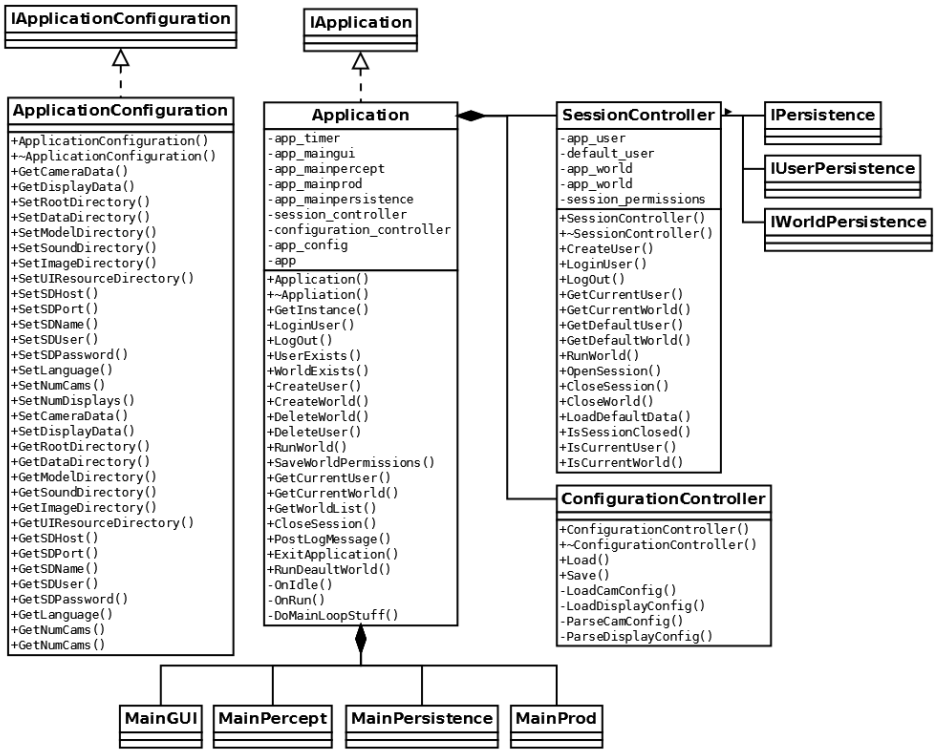


Figura 8.4: Diagrama de Clases UML, Segunda Demo. Módulo Principal de la Aplicación.

El módulo principal de la aplicación es el responsable de todas las acciones fundamentales del mismo. Es el responsable de instanciar y gestionar los módulos necesarios y suministrar métodos para la manipulación de la aplicación.

De esta forma, es ahora también responsable de la configuración de la aplicación, que realiza mediante el controlador ConfigurationController, así como de la sesión mediante SessionController.

SessionController da capacidades para cargar los datos de un usuario y permitir su entrada en el sistema, así como su salida, además de capacidades para saber cuál es el usuario y escenario actuales, o si un determinado usuario o escenario es el actual. Se entiende como sesión el estado en el cual un usuario que accede al sistema carga de un escenario para comenzar el uso del mismo. Esto es, se establece una sesión cuando se asigna un usuario y un escenario en el sistema.

Por su lado, ConfigurationController es usado fundamentalmente al comienzo y final de la aplicación para cargar y guardar datos relativos al funcionamiento de la misma. Por ello, sus métodos principales son Load y Save.

Diseño en UML - Módulo de Persistencia

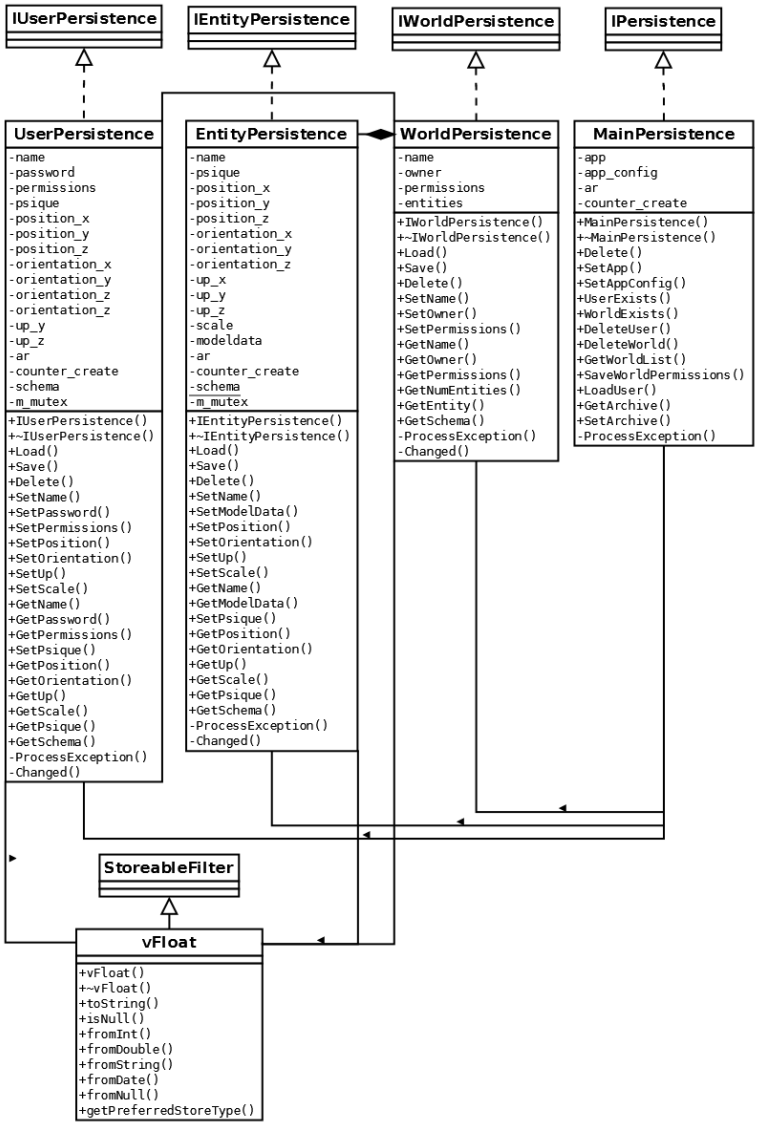
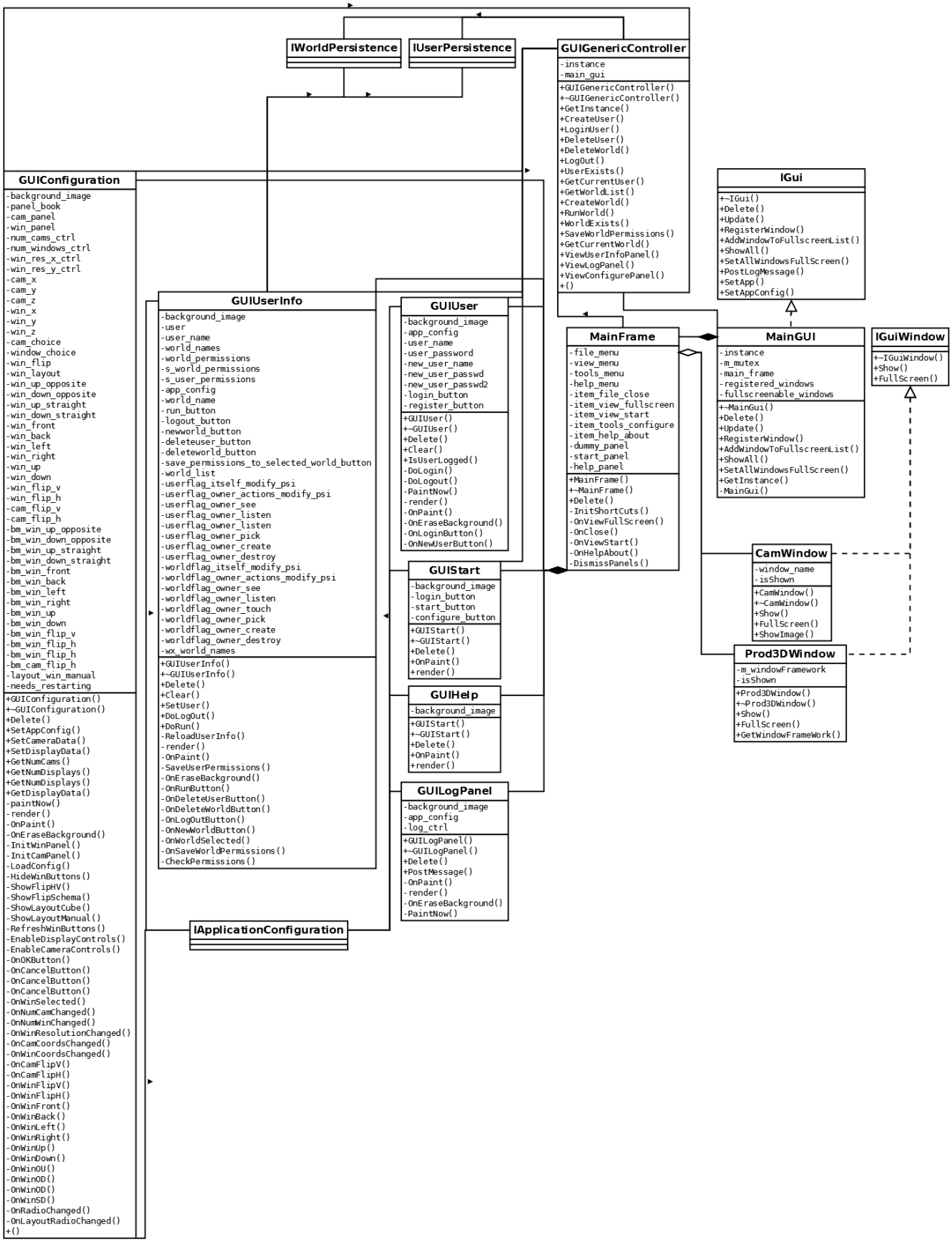


Figura 8.5: Diagrama de Clases UML, Segunda Demo. Módulo de Persistencia.

En este módulo se implementan las interfaces definidas referentes a persistencia en el núcleo de la interfaz y son un reflejo del mismo adaptado a la implementación final. Como se ha comentado se ha optado por fusionar los conceptos del modelo de objetos y modelo de persistencia. Siendo así la gestión de la persistencia de un objeto recae sobre él mismo (pe. para crear un objeto basta instanciarlo, y para guardar sus datos basta con llamar a su método 'Save'). Esta fusión podrá verse en mayor detalle en la sección de Implementación.

Como con el resto de módulos independientes existe un núcleo principal que será el encargado de iniciar el dispositivo de almacenamiento y de atender peticiones de otros módulos. Estas peticiones son consultas que pueden hacerse sin necesidad de instanciar el objeto, por ejemplo, para saber si un usuario determinado o un mundo existe u obtener la lista de mundos que un usuario puede ver.

Diseño en UML - Módulo de GUI



Una de las primeras consecuencias lógicas con la entrada de la capacidad para gestionar la configuración de la aplicación es su impacto sobre la interfaz gráfica de usuario. Este impacto se debe, por un lado, a la necesidad de localizar recursos estéticos como las imágenes de los botones, siendo el caso más común y básicamente el que sucede sobre las clases previamente existentes. Por otro lado, es también a través de la interfaz de usuario como éste va a editar los parámetros de configuración del sistema, mediante el panel de configuración. Además puede verse cambios relativos al uso del sistema, como son los paneles de login, de gestión de la información del usuario, y de log del sistema.

De esta forma, tenemos por un lado el panel GUIConfiguration. Este panel permitirá al usuario cambiar la cantidad de cámaras de las que se alimentará el módulo de Percepción, así como los parámetros de las mismas. De igual forma, podrá establecer la cantidad de displays que se mostrarán y su configuración, administrando vistas independientes de la escena. Gracias a esta posibilidad se dota de una gran flexibilidad al sistema para establecer modelos de instalaciones distintos: posibilidad de vision estéreo, frontal-cenital, etc. o espacios de escritorio, proyecciones frontales o sistemas Cave que envuelven al usuario con pantallas.

Por otro lado, tenemos el panel GUIUser, desde el cual un usuario podrá hacer login en el sistema o registrarse en el mismo como nuevo usuario. Este panel está estrechamente relacionado con GUIUSerInfo, que muestra al usuario actual la lista de mundos que ha creado y donde le permite modificar sus atributos, así como crear nuevos usuarios, destruirlos o borrarse él mismo del sistema.

Finalmente, se ha añadido un panel de log en el cual se mostrará información de interés del funcionamiento del sistema.

Hay que tener en cuenta que estas interfaces son sólo componentes que muestran y capturan información. No contienen ninguna lógica relevante de aplicación. Para estas tareas se hace uso, como es lógico, de un controlador. Sin embargo, las necesidades son mínimas y en su mayor parte este controlador sólo derivará las acciones hacia sus responsables. Por ello y por simplicidad, se ha decidido usar un controlador genérico para todo el módulo. Este controlador responderá a las necesidades de gestión de los paneles, para mostrar unos u otros. Pero a su vez, también encauzará las peticiones del usuario al interactuar con la interfaz, hacia el responsable de su gestión (pe. para crear un usuario nuevo o cargar un escenario).

Diseño en UML - Módulo de Producción

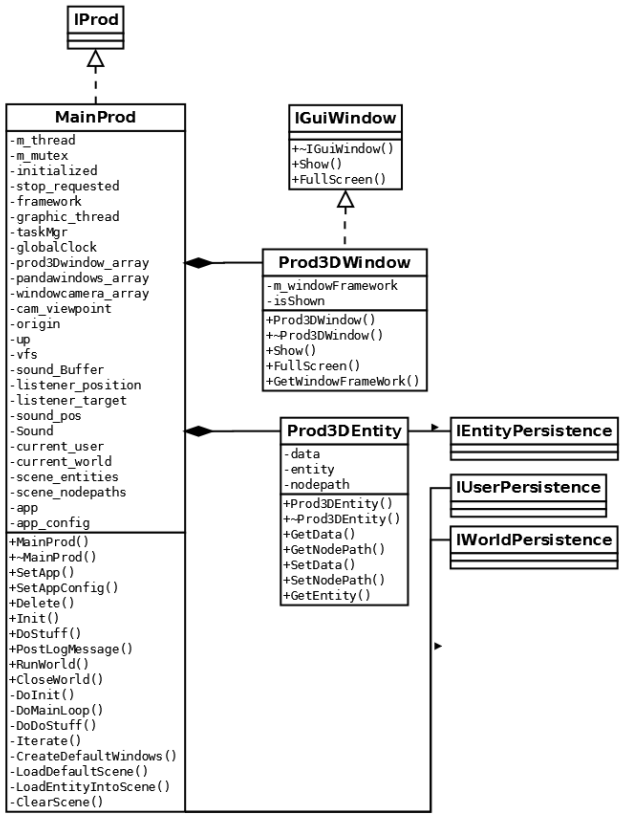


Figura 8.7: Diagrama de Clases UML, Segunda Demo. Módulo de Producción.

Durante esta fase los cambios en el Módulo de Producción son moderados. Por lado, usará la información de los parámetros de configuración para los displays que mostrará; por otro, se le ha añadido capacidad para limpiar y cargar datos en la escena. Estos datos serán los relativos al mundo que el usuario desee cargar.

De esta forma, destacan los métodos RunWorld() y CloseWorld(), que cargan un escenario y libera la escena respectivamente. También LoadEntityIntoScene() que carga una entidad persistida dentro de la escena, o LoadDefaultScene() que carga una escena sencilla por defecto, que se usará con carácter estético cuando no hay cargado ningún escenario.

Por otro lado, se puede ver que aparece una nueva clase llamada Prod3DEntity. Esta nueva clase representa las entidades 3D cargadas en la escena, que efectivamente se corresponderán con las Entidades persistidas que forman un escenario. Prod3Dentity encapsula los datos de Entity-Persistence y añade información concreta dependiente de la implementación, relacioanda con el motor gráfico. En este punto, esta información es concretamente el nodo del grafo de la escena.

Diseño en UML - Módulo de Percepción

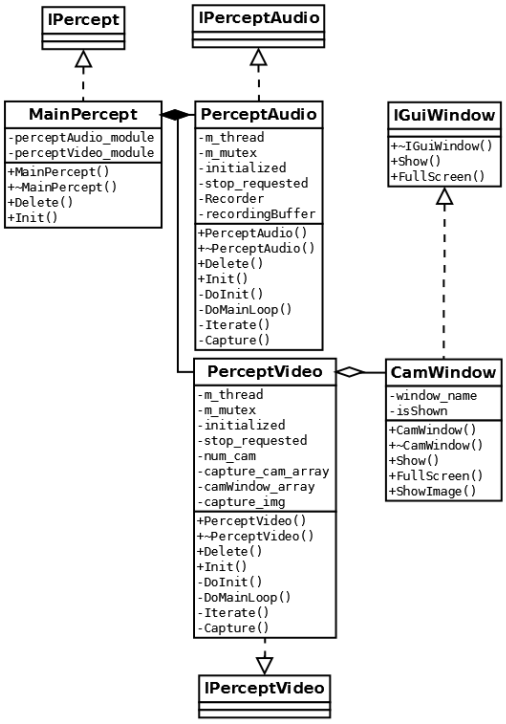


Figura 8.8: Diagrama de Clases UML, Segunda Demo. Módulo de Percepción.

Al igual que en el caso del módulo de Producción, este módulo sufre pocos cambios. En realidad incluso menos, ya que en esta fase del desarrollo no se abordarán funcionalidades nuevas en la percepción. Sin embargo, el módulo sí se ve afectado por los parámetros de configuración de la aplicación, permitiendo establecer el número de cámaras desde las que se capturan imágenes y los parámetros de las mismas.

8.3. Implementación

Los módulos afectados en la presente fase son los siguiente:

- core: Añadida interfaz de configuración de aplicación IApplicationConfiguration.
- igui: añadidas interfaces gráficas para la gestión de usuarios, escenas y paneles de configuración de la aplicación.
- ipercept: Configuración del módulo.
- ipersistence: Persistencia de los objetos usuario, mundo y entidad.

- iprod: Configuración del módulo.
- vox: Controlador de la configuración de la Aplicación.

Núcleo de la Interfaz

\src\core

- IApplicationConfiguration: Interfaz destinada a la configuración de la aplicación y que puede ser usada por parte de los módulos con independencia de la implementación final.
  Algunas características son, sin embargo, inherentes a la naturaleza de los módulos y los dispositivos genéricos. Hablamos del caso de las propiedades clásicas de los dispositivos de entrada como son las cámaras y de salida como las ventanas de visualización. Algunas de estas opciones son la resolución y orientación de los mismos. De esta forma, se añaden las clases CameraData y DisplayData.

```
class DisplayData
{ public:
   DisplayData() : x(0),y(0),z(0),flip_h(false),flip_v(false),
      resolution_x(800),resolution_y(600){}
   ~DisplayData() {}
   double x, y, z;
   bool flip_h, flip_v;
   unsigned int resolution_x, resolution_y; };
```

  Finalmente, se añaden los métodos a utilizar para acceder y modificar las opciones de configuración de la aplicación. Se muestran algunas de las principales. Para ver más detalles consultar la sección B.3, página 112.

```
class IApplicationConfiguration
{ public:
   virtual unsigned int GetNumCams()     = 0;
   virtual unsigned int GetNumDisplays() = 0;
   virtual core::CameraData  GetCameraData(const int &id)=0;
   virtual core::DisplayData GetDisplayData(const int &id)=0;
   virtual std::string GetSDHost() = 0;
   virtual std::string GetSDPort() = 0;
   virtual std::string GetSDPassword()          = 0;
   virtual std::string GetUIResourceDirectory() = 0;
```

Aplicación Principal

\apps\vox

Se añade la clase ApplicationConfiguration que representa el modelo de datos de la configuración del sistema, y los controladores ConfigurationController y SessionController. Se describen a continuación:

- \apps\ApplicationConfiguration: Implementa la interfaz IApplicationConfiguration que peude verse en el apartado anterior, disponiendo de los mismo métodos para obtener y asignar valores.

- \apps\controllers\ConfigurationController: Controllador destinado a la gestión de la configuración. Carga y guarda los parámetros mediate los métodos Load() y Save() de forma estructurada en un archivo de configuración en texto plano llamado config.ini. Como puede verse, para esta tarea se buscan las etiquetas asociadas a cada atributo y toma o guarda directamente su valor.

```
bool ConfigurationController::Load()
{   if ( config_file.is_open() )
    { while (!config_file.eof())
      { if ( tag == "[DATA_DIRECTORY]" )
        { config_file.getline(str, size);
          app_config->SetDataDirectory(str); }
```

Hay que tener en cuenta que la mayoría de los datos son atómicos, es decir, un atributo está asociado sólo con un único valor indivisible. Sin embargo, existen atributos compuestos de múltiples valores que requieren parsing para su lectura y escritura. Este es el caso de la configuración particular de las distintas cámaras y displays. Para ellos se usan etiquetas a modo de separadores para distinguir cada dispositivo y sus atributos.

```
std::string ConfigurationController::ParseDisplayConfig()
{ ...
  unsigned int win_num = app_config->GetNumDisplays();
  for (unsigned int i = 1; i <= win_num; i++)
  { core::DisplayData data = app_config->GetDisplayData(i);
    wop << "::ID::" << i
    << "::X::" << data.x << "::Y::" << data.y << "::Z::" << data.z
    << "::FLIPH::" << data.flip_h << "::FLIPV::" << data.flip_v
    << "::RES_X::" << data.resolution_x
    << "::RES_Y::" << data.resolution_y;        }
  ... }
```

Módulo de Persistencia

\src\ipersistence

La clase principal del Módulo de Persistencia y que implementa la interfaz IPersistence es MainPersistence. Esta clase se encarga de la inicialización del módulo y del dispositivo de almacenamiento. La verdadera intención era hacer uso de una libería que permitiera hacer uso de un esquema de persistencia Orientado a Objetos. Sin embargo, no fue posible encontrar una que cumpliera los requisitos software de la aplicación. Por ello, se decidió adoptar una solución intermedia. De esta forma se decide utilizar una base de datos relacional, PostGreSQL, y un mapper objeto-relacional, Debea. Para ver los detalles de implementación necesarios para configurar el módulo con estas librerías se puede consultar la sección ?? página ??. De esta forma, el mapper permite fusionar el modelo de objeto y el modelo de persistencia en uno. Sin embargo, es interesante saber que, en el fondo, al tratarse el dispositivo de almacenamiento de una base de datos SQL, también se permiten realizar consultas de forma manual.

- \apps\MainPersistence: Inicializa la conexión con la base de datos y en caso de no existir contenido, es la responsable de crear las tablas de las entidades e introducir datos por defecto. Para crear las tablas, solicita los esquemas de traducción a las clases de los objetos a persistir.

```
1  MainPersistence::MainPersistence(IApplicationConfiguration *app_cfg)
2  { std::string db_name = app_cfg->GetSDName();
3    std::string db_user = app_cfg->GetSDUser();
4    std::string db_passwd = app_cfg->GetSDPassword();
5    std::stringstream connect;
6    connect << "dbname=" << db_name << " user="
7            << db_user << " password=" << db_passwd;
8    ar.setIdFetcher(new dba::GenericFetcher());
9    unlink(db_name.c_str());
10   ar.open("dbapgsql-static", connect.str().c_str());
11   //create needed tables
12   ar.getOStream().sendUpdate(counter_create);
13   ar.getOStream().sendUpdate((*(UserPersistence::GetSchema())));
14   ar.getOStream().sendUpdate((*(EntityPersistence::GetSchema())));
15   ar.getOStream().sendUpdate((*(WorldPersistence::GetSchema())));
16   //first id //
17   ar.getOStream().sendUpdate(dba::SQL("INSERT INTO
        debea_object_count VALUES (:d)") << 1);
18 ...
```

Los modelos de los objetos están ligados directamente a su persistencia, por lo que MainPersistence no tiene responsabilidad explícita sobre la gestión particular de cada uno de ellos. Sin embargo, dispone de mecanismos para consultar la base de datos y obtener información de interés, como puede ser consultar si un usuario o un mundo existe, o recuperar la lista de mundos de los que un usuario es propietario. También permite borrar un escenario o un usuario, sin necesidad de cargarlo en el sistema, o modificar las propiedades de un escenario.

- \apps\UserPersistence: Representa al objeto usuario y contiene sus datos y métodos para

guardarlos y recuperarlos. Como puede verse, la clase hereda de dba::Storeable lo que permitirá hacer uso de los mecanismos de persistencia. Aunque el mapper tiene una API suficiente posee algunas carencias. Por un lado es necesario establecer el esquema de la clase (definición de la tabla en la base de datos) y la relación de las variables con las mismas.

```
//::MAPPING:: Class name, parent class name and relation name
BEGIN_STORE_TABLE(UserPersistence, dba::Storeable, "user_table")
 BIND_STR(UserPersistence::name,     dba::String, "name"    )
 BIND_STR(UserPersistence::password, dba::String, "password")
 BIND_INT(UserPersistence::psique,   dba::Int,    "psique"  )
 ...
END_STORE_TABLE()
//SQL schema
dba::SQL UserPersistence::schema(
"CREATE TABLE user_table ("
"  id INT PRIMARY KEY,"
"  name VARCHAR,"
"  password VARCHAR,"
"  psique INT,"
...
```

Una vez mapeado se pueden crear instancias de la clase que ya tienen capacidad directa de persistencia. Sin embargo, se desea una abstracción completa de la implementación de estos mecanismos, además de un mejor diseño para ofrecer una mejor interfaz de uso. Para ello, se encapsulan los métodos básicos de carga, almacenamiento y borrado en los métodos Load(), Save() y Delete().

```
bool UserPersistence::Load(const int &id)
{ try
  { boost::mutex::scoped_lock lock(m_mutex);
    UserPersistence new_object;
    dba::SQLIStream istream = ar->getIStream();
    istream.setWhereId(id);
    bool success = istream.get(&new_object);
    this->operator =(new_object);
    return success; }
  catch (const dba::SQLException& pEx){ ProcessException(pEx); }
  catch (const dba::Exception& pEx)   { ProcessException(pEx); }
  return false;
}
void UserPersistence::Save()
{ try
  { boost::mutex::scoped_lock lock(m_mutex);
    ar->getOStream().put(this); }
  catch (const dba::SQLException& pEx){ ProcessException(pEx); }
```

```
19    catch (const dba::Exception& pEx)   { ProcessException(pEx); }
20  }
21  void UserPersistence::Delete()
22  { try
23    { boost::mutex::scoped_lock lock(m_mutex);
24      this->setState(dba::Storeable::stState(DELETED));
25      ar->getOStream().put(this); }
26    catch (const dba::SQLException& pEx){ ProcessException(pEx); }
27    catch (const dba::Exception& pEx)   { ProcessException(pEx); }
28  }
```

Hay que tener en cuenta que los objetos puedes ser accedidos desde distintos módulos que se ejecutan en hilos en paralelo, por lo que es necesario el uso de cerrojos para sincronizar los accesos. Se utiliza para ello los cerrojos de ámbito que ofrece boost.

- \apps\EntityPersistence: Es la clase que implementa la interfaz IEntityPersistence que modela el objeto Entidad. Una entidad es un elemento básico de los cuales se compone una escena. Su implementación sigue exactamente el mismo esquema que el caso anterior: un mapeo de variables, una definición de esquema, y métodos que encapsulan la API de Debea para las operaciones de persistencia.

  Otro detalle a tener en cuenta es la necesidad de un filtro para mapear las variables de tipo Float o Double correctamente, ya que la librería demostraba problemas en sistemas de localización no inglesa, debido a los separadores usados como punto decimal. Estudiado el problema se corrigió y se añadió un filtro adecuado vFloat. Los cambios fueron remitidos y pendientes de admisión en revisión.

  Finalmente, para reflejar la relación de pertenencia que tienen los escenarios con las entidades, es necesario incluir en el esquema de la tabla de la base de datos una entrada que es la clave ajena de mundo al que pertence.

```
1   //::MAPPING:: Class name, parent class name and relation name
2   BEGIN_STORE_TABLE(EntityPersistence, dba::Storeable, "entity_table")
3    BIND_STR(EntityPersistence::name,       dba::String, "name"       )
4    BIND_FLT(EntityPersistence::position_x, dba::vFloat, "position_x" )
5   END_STORE_TABLE()
6   //SQL schema
7   dba::SQL EntityPersistence::schema(
8   "CREATE TABLE entity_table ("
9   "  id INT PRIMARY KEY,"
10  "  name VARCHAR,"
11  "  position_x FLOAT,"
12  "  fk_world INT",
```

```
13 ...
```

- \apps\WorldPersistence: El último modelo de objeto a persistir es el Mundo o escenario. Simplemente define una colección de Entidades y algunos atributos como nombre, permisos o el usuario que creó dicho mundo.

  Para mapear la lista de entidades que contiene un Mundo se usa la macro BIND_COL, que establece una relación de cada elemento con la tabla correspondiente y que implica la necesidad de que exista una clave ajena en el esquema de la otra clase.

  Puede apreciarse que existe una relación similar entre Usuario y Mundo, pero en ese caso no se lleva a cabo por dos motivos: Primero, porque conceptualmente el objeto usuario no se compone de Mundos, y segundo porque, aunque en el desarrollo de este proyecto un usuario sólo puede acceder a los mundos que ha creado, se mantiene una perspectiva abierta para trabajo futuro en la que otros usuarios conectados puedan acceder o compartir cualquier mundo, siendo sus acciones limitadas por los permisos que el creador definiera para el mismo.

```
1  //::MAPPING:: Class name, parent class name and relation name
2  BEGIN_STORE_TABLE(WorldPersistence, dba::Storeable, "world_table")
3   BIND_STR (WorldPersistence::name,        dba::String, "name")
4   BIND_STR (WorldPersistence::owner,       dba::String, "owner")
5   BIND_INT (WorldPersistence::permissions, dba::Int,    "permissions"
       )
6   BIND_COL (WorldPersistence::entities,
7             dba::stdList<EntityPersistence>, "fk_world")
8  END_STORE_TABLE()
9  //SQL schema
10 dba::SQL WorldPersistence::schema(
11 "CREATE TABLE world_table ("
12 "  id INT PRIMARY KEY,"
13 "  name VARCHAR,"
14 "  owner VARCHAR,"
15 "  permissions INT"
```

Módulo de GUI

\src\igui

- \src\igui\controllers\GUIGenericController: Dado que la lógica asociada al módulo de GUI es simple y con pocas opciones, que además son comunes, se ha incluido un controlador genérico. Este controlador es utilizado para alternar entre la visualización de distintos paneles de GUI, así como para gestionar los accesos a funcionalidades del núcleo principal del módulo, maingui.

```
class GUIGenericController
{public:
  static GUIGenericController* GetInstance() {return instance;}
  bool CreateUser(const string &name,const string &passwd);
  bool LoginUser(const string &name,const string &passwd);
  bool DeleteUser(const string &name);
  bool DeleteWorld(const string &name);
  void LogOut();
  void ViewUserInfoPanel();
  void ViewLogPanel();
  void ViewConfigurePanel();
  private:
  static GUIGenericController *instance;
  ... };
```

- \src\igui\GUIUser: Interfaz destinada al acceso de los usuarios, mediante nombre y contraseña, y para la creación de nuevas cuentas de usuario. La lógica relacionada al login y a la creación de nuevos usuarios se realiza a través del controlador GUIGenericController. Sin embargo, si las acciones no son responsabilidad de este módulo lo único que hace GUIGenericController es derivarlas al responsable.

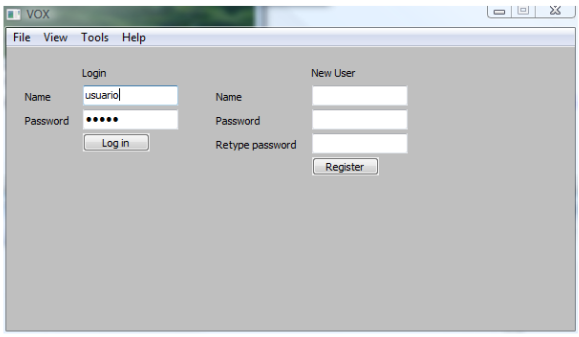


Figura 8.9: GUI: User login

El código de la interfaz es simple y común a las mostradas anteriormente. De forma ilustrativa se muestra el caso de la operación de login. Primero, Se recuperan los valores introducidos por el usuario en los controles, luego se usa el controlador, que relegará la consulta. En caso de no tener éxito la petición se muestran los mensajes de error.

```
BEGIN_EVENT_TABLE(GUIUser, wxPanel)
   EVT_PAINT   (                  OnPaint          )
   EVT_BUTTON  ( wxID_LOGIN     , OnLoginButton    )
   EVT_BUTTON  ( wxID_REGISTER  , OnNewUserButton  )
```

```
5  END_EVENT_TABLE()
6
7  void GUIUser::OnLoginButton(wxCommandEvent& WXUNUSED(event))
8  { s_user_name = user_name->GetValue();
9    s_user_password = user_passwd->GetValue();
10   DoLogin(s_user_name, s_user_password);   }
11
12 void GUIUser::DoLogin(const string &name, const string &passwd)
13 { GUIGenericController *guiGc = GUIGenericController::GetInstance();
14   if (guiGc != NULL) login = guiGc->LoginUser(name, passwd);
15   if (login)
16   { user_logged_in = true;
17     guiGc->ViewUserInfoPanel(); }
18   else
19   { user_logged_in = false;
20     wxMessageDialog message_dialog(this, _("User or password
         incorrect"), "Message box", wxOK \| wxICON_EXCLAMATION \|
         wxSTAY_ON_TOP);
21     message_dialog.ShowModal(); } }
```

- \src\igui\GUIUserInfo: Interfaz destinada a la gestión del usuario y de sus datos. Desde este panel el usuario puede ver la lista de Entornos que tiene, crear nuevos entornos, borrarlos, modificar sus permisos o lanzarlos para su ejecución en la sesión. Estas acciones se realizan a través del controlador GUIGenericController.

Como se comentaba en el punto anterior, no es el controlador GUIGenericController el que realiza directamente las acciones de login en el sistema, y tampoco el núcleo principal del módulo de gui. De hecho, sólo derivan las peticiones a los módulos responsables. En este caso, la responsabilidad de gestionar la sesión, así como todas las acciones sobre el sistema caen sobre el módulo principal de la Aplicación, que en este caso, a su vez, relegará en en el controlador de la sesión SessionController, que hará uso finalmente del módulo de Persistencia. Para más detalles a este respecto puede verse ??

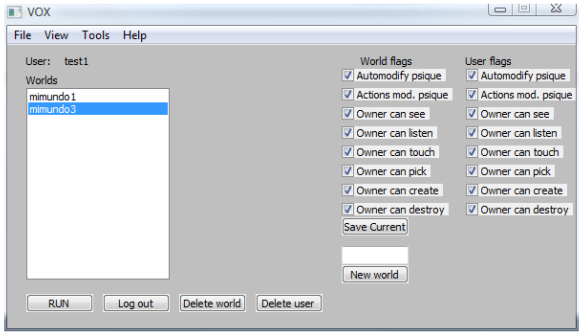


Figura 8.10: GUI: User info

- \src\igui\GUIConfiguration: Este panel de la interfaz está destinado a la configuración de la aplicación. En el puede establecerse el número de cámaras que se van a conectar y el número de ventantas de render que se quieren ver. Además por cada cámara o display se pueden establecer parámetros de configuración, como puede ser su orientación o resolución. Debido a limitaciones de las librerías y a la relevancia de algunos cambios, algunos de éstos requieren el reinicio de la aplicación y de tal forma es notificado al usuario. Por otro lado, hay que tener en cuenta que algunos parámetros, como el calibrado posicional de las cámaras en el espacio, no son usados todavía aunque ya se almacenan y se permite su gestión.

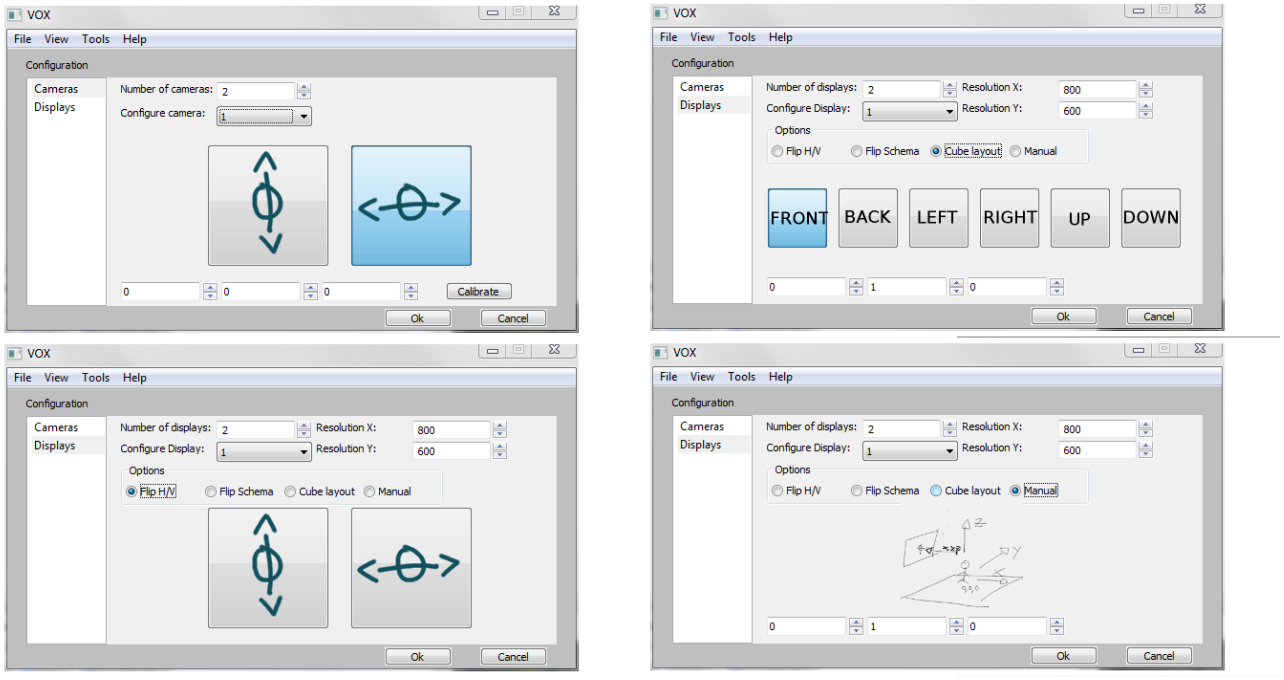


Figura 8.11: GUI: Panel de configuración

- \src\igui\GUILogPanel: La última de las interfaces añadidas muestra una ventana donde aparecerán mensajes que puedan resultar de interés enviados desde cualquier módulo de la aplicación. No pretende ser un log exhaustivo del funcionamiento de la aplicación sino una forma de ofrecer información a modo de curiosidad. En el gráfico pueden verse mensajes de incialización, así como la entrada y salida de usuarios en el sistema.

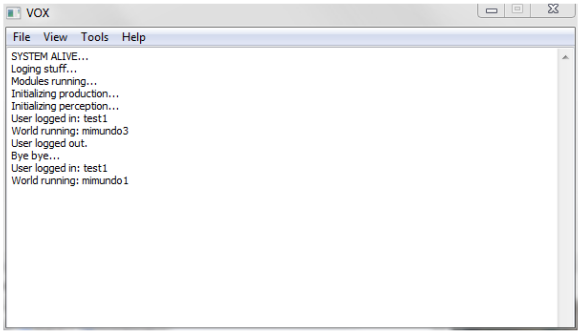


Figura 8.12: GUI: Panel de log

Módulo de Producción

\src\iprod En esta fase el módulo de Producción sufre pocos cambios. Se añade la clase Prod3DEntity que encapsula el objeto entidad añadiendo particularidades propias del motor gráfico y métodos a MainProd que permiten cargar y limpiar la escena.

- \src\iprod\Prod3DEntity: Como se ha comentado encapsula el objeto EntityPersistence y añade particularidades relacionadas con la implementación del módulo de aplicación. En este caso se añade un puntero al nodo del grafo de la escena que se corresponde con el objeto 3D asociando.

```
1  class Prod3DEntity
2  { public:
3    Prod3DEntity(core::IEntityPersistence* ent);
4    virtual ~Prod3DEntity();
5    std::string GetData()               { return data;     }
6    NodePath*     GetNodePath()             { return nodepath; }
7    core::IEntityPersistence* GetEntity(){return entity;   }
8    void SetData(const std::string &value){data = value;   }
9    void SetNodePath(NodePath *value);
10   private:
11   std::string data;
12   core::IEntityPersistence* entity;
13   NodePath *nodepath; };
14
15 Prod3DEntity::Prod3DEntity(core::IEntityPersistence* ent)
```

```
16 { if (entity != NULL ) data = entity->GetModelData(); }
17
18 void Prod3DEntity::SetNodePath(NodePath *value)
19 { nodepath = value;
20   if ( (entity != NULL) && (nodepath != NULL) )
21   { float posx, posy, posz, rotx, roty, rotz, scale;
22     entity->GetPosition(posx, posy, posz);
23     entity->GetOrientation(rotx, roty, rotz);
24     entity->GetScale(scale);
25     nodepath->set_pos(posx,posy,posz);
26     nodepath->set_hpr(rotx, roty, rotz);
27     nodepath->set_scale(scale,scale,scale);     } }
```

- \src\iprod\MainProd: Durante esta fase el módulo de Producción sufre algunos cambios leves, aunque ha tenido que hacerse frente a ciertas dificultades del motor gráfico. Se hace uso de la configuración de la aplicación para mostrar las ventanas de render y se dota de capacidad para cargar y cerrar escenarios. Por ello, cabe destacar los métodos RunWorld(), que lee los datos de un objeto Mundo, extrae sus entidades y las carga en escena, y CloseWorld(), que vacía la escena y limpia los datos internos.

```
1  void MainProd::CloseWorld()
2  { ClearScene();
3    { boost::mutex::scoped_lock lock(m_mutex);
4      current_user  = NULL;
5      current_world = NULL;
6      for (unsigned int i = 0; i < scene_entities.size(); i++)
7       delete scene_entities[i];
8      scene_entities.clear(); } }
9  bool MainProd::RunWorld(core::IUserPersistence *user, core::
      IWorldPersistence *world)
10 { boost::mutex::scoped_lock lock(m_mutex);
11   current_user  = user;
12   current_world = world;
13   for (int i=0; i < current_world->GetNumEntities(); i++)
14   { IEntityPersistence *ient = current_world->GetEntity(i);
15     Prod3DEntity *new_entity = new Prod3DEntity(ient);
16     scene_entities.push_back(new_entity);
17     LoadEntityIntoScene(new_entity); }
18   initialized = true;
```

Dentro de RunWorld es usado el método LoadEntityIntoScene() que es usado para cargar las entidades leídas en la escena 3D. Se recuerda que por motivos de eficiencia el modelo 3D leído es cargado sólo en el primer framework, correspondiente a la primera vista, y posteriormente se instancian en el resto.

```
void MainProd::LoadEntityIntoScene(Prod3DEntity * entity)
{ std::string data = entity->GetData();
  scene_nodepaths[entity] = pandawindows_array[1]->load_model(
     framework.get_models(),data);
  entity->SetNodePath(&(scene_nodepaths[entity]));
  scene_nodepaths[entity].reparent_to(pandawindows_array[1]->
     get_render());
  std::map<int, WindowFramework*>::iterator iter =
     pandawindows_array.begin(); iter++;
  while(iter != pandawindows_array.end())
  { scene_nodepaths[entity].instance_to(iter->second->get_render());
    iter++; } }
```

Las últimas modificaciones afectan a la configuración de las ventanas de render. Por un lado se cargan los valores de la resolución de cada ventana de render y por otro se realiza una transformación horizontal o vertical de la imagen renderizada. De hecho, por eficiencia lo que se hace es escalar la cámara por el valor -1 en los ejes horizontal o vertical según corresponda. De esta forma no será necesario transformar la imagen después de ser renderizada. Un detalle a tener en cuenta es que al realizar el escalado en uno de los ejes del plano de render, el vector normal se invierte, por lo que la ocultación de caras usada en la escena (cull facing) también debe ser invertido o se verán las caras de los polígonos contrarias a las deseadas.

```
for (unsigned int i = 1; lock && (i <= num_windows); i++)
{ int flip_me_x = (app_config->GetDisplayData(i).flip_h) ? -1 : 1;
  int flip_me_y = (app_config->GetDisplayData(i).flip_v) ? -1 : 1;
  windowcamera_array[i].set_sx(flip_me_x);
  windowcamera_array[i].set_sz(flip_me_y);
  if ( (flip_me_x == -1) ^ (flip_me_y == -1) )
   pandawindows_array[i]->get_render().set_attrib(CullFaceAttrib::
      make_reverse());
  else
   pandawindows_array[i]->get_render().set_attrib(CullFaceAttrib::
      make_default()); } }
```

Módulo de Percepción

\src\ipercept\PerceptVideo: Los cambios en este módulo son mínimos y afectan sólamente al establecimiento de los parámetros de configuración. En este caso se aplica una transformación de reflejo horizontal o vertical según se haya especificado en la configuración.

```
if ( app_config != NULL )
{ flip_me_x = app_config->GetCameraData(iter->first).flip_h;
  flip_me_y = app_config->GetCameraData(iter->first).flip_v;
```

```
4     if(flip_me_x && !flip_me_y) //Flip Horizontally
5     {     IplImage *aux = capture_img;
6       cvFlip(capture_img, capture_img, 1); }
7     else if(!flip_me_x && flip_me_y) //Flip Vertically
8     { IplImage *aux = capture_img;
9       cvFlip(capture_img, capture_img, 0); }
10    else if(flip_me_x && flip_me_y) //Flip Both
11    { IplImage *aux = capture_img;
12      cvFlip(capture_img, capture_img, -1); }
```

8.4. Validación y Publicidad

8.4.1. Validación

- Comprobación de uso de recursos de la máquina mediante las herramientas del sistema. En la máquina en la que se desarrolla la aplicación muestra consumir un 4 y un 6 % de CPU, con 21 subprocesos asociados y 85Mb de memoria mantenidos con el escenario por defecto de prueba. Al cargar los distintos escenarios se ven cambios dependiendo de la complegidad de los mismos, lo cual es esperado.

- Comprobración de la ejecución y cierre correctos, sin salidas de la aplicación inesperadas ni memory leaks.

- Uso de herramientas para medición de frames por segundo para comprobar el rendimiento de la ventana de render. No se aprecian cambios en la tasa de frames por segundo que se mantiene a 60fps para dos ventanas de render. Teniendo en cuenta que 60 es el límite máximo impuesto por la sincronización vertical del monitor.

Se muestran algunas capturas del estado actual de la aplicación:

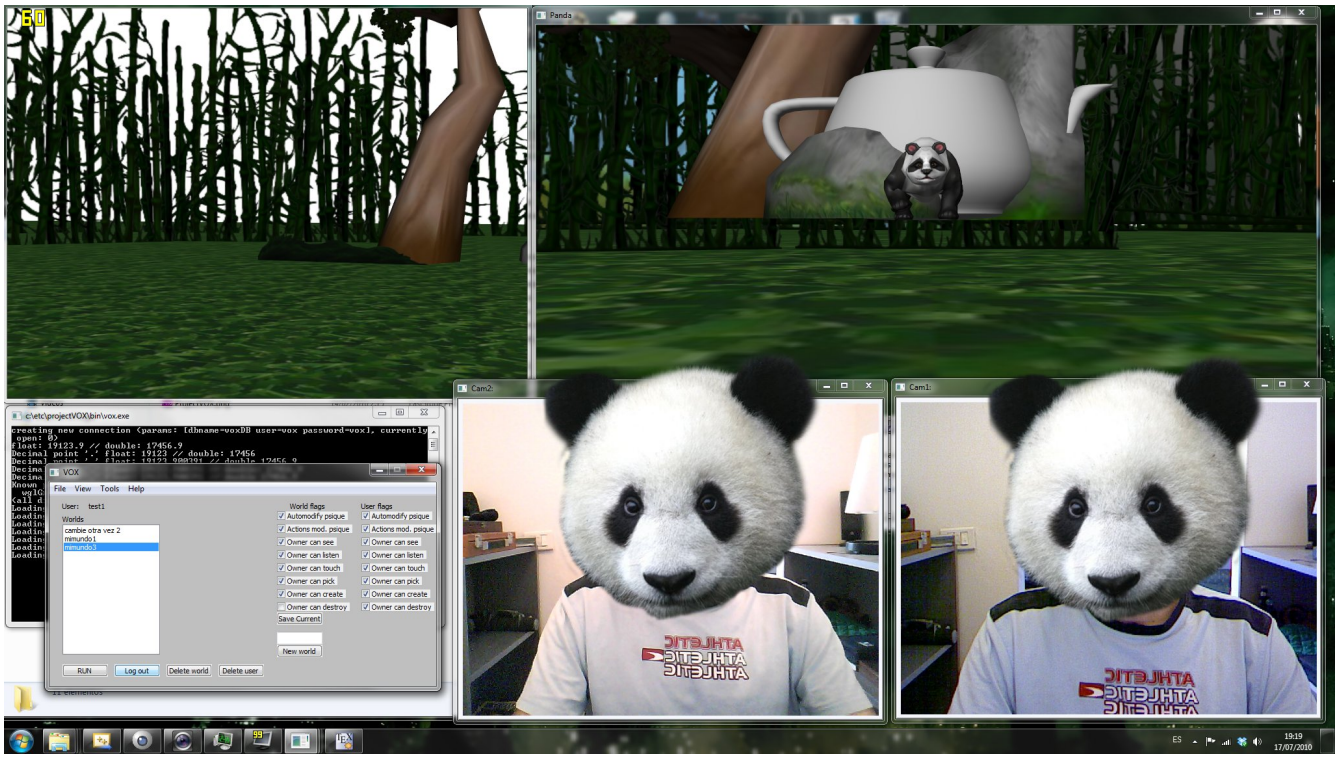


Figura 8.13: Fase 3: Segunda Demo

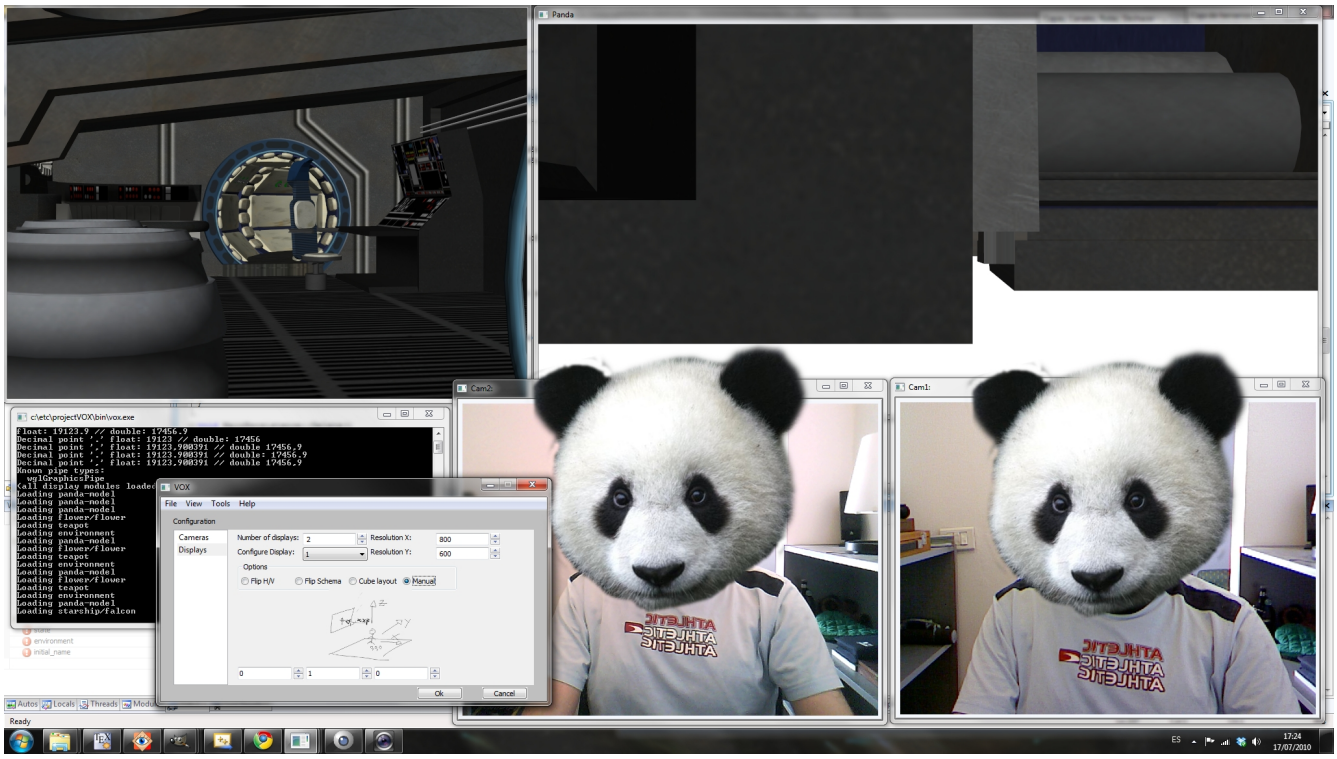


Figura 8.14: Fase 3: Segunda Demo

Capítulo 9

Etapa 3+i: Funcionalidades Añadidas

9.1. Análisis

9.2. Diseño

Diseño en UML - Resumen

Diseño en UML - Núcleo de la Interfaz

Diseño en UML - Módulo de Aplicación

Diseño en UML - Módulo de Persistencia

Diseño en UML - Módulo de GUI

Diseño en UML - Módulo de Producción

Diseño en UML - Módulo de Percepción

9.3. Implementación

9.4. Validación y Publicidad

Capítulo 10

Etapa n: Revisión Final

10.1. Implementación

10.2. Validación y Publicidad

Capítulo 11

Etapa n+1: Publicidad

11.1. Análisis

11.2. Diseño

11.3. Implementación

11.4. Validación y Publicidad

Capítulo 12

Etapa n+2: Implantación Final

12.1. Análisis

12.2. Diseño

12.3. Implementación

12.4. Validación y Publicidad

Capítulo 13

Resultados y conclusiones

Desarrollo de los resultados y conclusiones del proyecto. Se debe intentar resaltar el interés del proyecto y la calidad del trabajo realizado. Ha llegado el momento de "vender"nuestro trabajo. Se deben incluir aspectos como:

- Calidad, dificultad y amplitud del trabajo desarrollado que justifique el tiempo de dedicación al proyecto.
- Aspectos integradores de las disciplinas de la titulación de Ingeniero en Informática.
- Impacto social. Utilidad del proyecto en el ámbito social
- Facilidad de utilización de los resultados del proyecto por terceras personas.
- Publicidad de los resultados del proyecto a través de páginas web, etc.. Cuando de los resultados del proyecto se derive un prototipo o programa de utilización se debe poner a disposición del público en general una versión de demostración de dicho prototipo.
- Cualquier otro mérito.

Capítulo 14

Trabajo Futuro

Continuidad del trabajo realizado a través de una implementación, o utilidad real del proyecto, o a través de otros proyectos fin de carrera.
-Esquema multiusuario online. -Base de Datos Orientada a Objetos, Tiempo Real.

Apéndice A

Manuales de usuario

En el caso de que el desarrollo (y/o naturaleza del proyecto haya dado lugar a la creación de manuales de usuario, habrá que ponerlo aquí).

Apéndice B

Detalles técnicos sobre la implementación del proyecto

En este apéndice se detallan los aspectos técnicos relacionados con la implementación del proyecto de forma más concreta.

B.1. Incompatibilidades

Uno de los aspectos a tener en cuenta es que al usar una cantidad considerable de librerías externas aparecen con frecuencia incompatibilidades que deben ser resueltas. Algunas de estas incompatibilidades pueden afectar sólo a los proyectos de los que dependa un módulo concreto o a la totalidad del proyecto.

B.1.1. winsock

Uno de los problemas más comunes fue la incompatibilidad entre librerías relacionado con librerías del sistema operativo, como fue el caso con winsock. Varias de las librerías usan módulos relacionados incompatibles, por ejemplo Panda3D utiliza winsock, mientras OpenCV usa ws2def, ambas son incompatibles entre sí. Para resolver estos problemas se añaden al inicio de la cabecera de la aplicación las siguientes líneas:

```
#ifdef _WINDOWS
#include <winsock2.h>
#endif
```

Estos problemas están ligados a las librerías del sistema operativo que utilizan las librerías externas y la inclusión sólo es necesaria para el sistema operativo Windows y se realizaría la inclusión si la macro está definida.

B.1.2. CLR

Otra de las incompatibilidades más comunes fue la necesidad de desactivar el CLR, Common language runtime support. Debe ser puesto a No Common language runtime support.

B.1.3. NDEBUG

Panda3D presenta graves problemas si se utiliza la macro _NDEBUG en Release, por lo que debe quitarse de todo el proyecto.

B.2. Third Parties

Todas las librerías externas utilizadas se encuentran dentro de la rama *backslash*extern.

B.2.1. wxWidgets

Las instrucciones para descargar y compilar wxWidgets pueden encontrarse en la web oficial [24]. La distribución viene proyectos preparados para Visual Studio 2008. Sólo es necesario descargar la distribución, abrir la solución y compilarlo en Debug y Release, para obtener las librerías, sin necesidad de ningún ajuste en la compilación.

B.2.2. Boost

La integración de Boost en el proyecto dio algunos problemas. Se obtuvo la distribución a partir del svn y se siguieron los pasos descritos en la documentación. Sin embargo, para poder añadir correctamente la librería a la solución son necesarias algunas opciones extras al compilarla. Estas opciones son la especificación de multi-threading y el uso de librerías estáticas. El comando final para compilar la librería se muestra a continuación:

```
>.\bjam link=static threading=multi threading=single variant=release variant=debug runtime-link=
    static
```

Hay que tener en cuenta que es necesario desactivar el CLR para evitar incompatibilidades.

B.2.3. SFML

Puede obtenerse desde svn o descargando algunos de los paquetes que existen, con proyectos preparados para distintos IDEs. La compilación es sencilla y directa.

Sin embargo, es importante destacar que la documentación acerca del paquete de sonido es confusa y en ocasiones errónea. Sin embargo, es útil para conocer las herramientas que dispone.

Se describen los detalles a tener en cuenta, especialmente en relación a la localización del espacio 3D. En concreto:

- El eje X es creciente hacia la derecha.
- El eje Y es creciente hacia arriba.
- El eje Z es creciente y apunta hacia afuera de la pantalla, hacia el usuario.
- El método SetTarget(), para orientar el oyente en una dirección, usa un vector normalizado relativo al oyente, no el punto del objeto al que escucha, como puede malinterpretarse por la documentación.
- La librería NO permite la correcta espacialización del oyente en un entorno 3D tal y como se distribuye. Esto es debido a que enmascara la creación del oyente de OpenAL, ignorando una componente crucial: el vector de verticalidad. Establecen este vector siempre como vertical en el espacio absoluto. Como puede apreciarse en la figura B.1, esto impide la correcta orientación del oyente e impide que se pueda distinguir la orientación derecha-izquierda de la 'cabeza'. Sólo sería válido para posicionamiento 2D o 3D donde el vector de dirección del usuario se moviera en el plano horizontal y la verticalidad del oyente no cambiara.

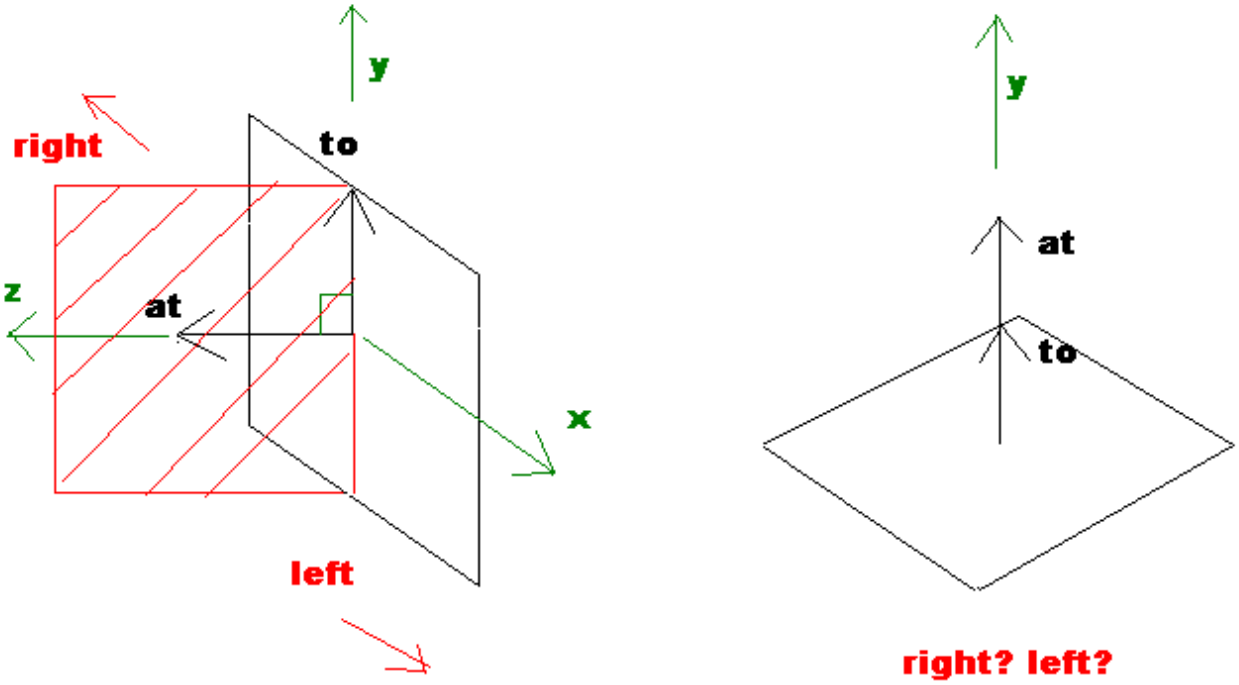


Figura B.1: Necesidad del vector de verticalidad.

Esto puede verse claramente en la Definición del método de la clase Listener:

```
/// Change the orientation of the listener (the point he must look at) (
   take 3 values)
void Listener::SetTarget(float X, float Y, float Z)
{   float Orientation[] = {X, Y, Z, 0.f, 1.f, 0.f};
    ALCheck(alListenerfv(AL_ORIENTATION, Orientation)); }
```

Sin embargo, el arreglo es sencillo. Basta con añadir a la clase Listener la siguiente función:

```
void Listener::SetTarget(float X, float Y, float Z, float u, float v,
   float p)
{   float Orientation[] = {X, Y, Z, u, v, p};
    ALCheck(alListenerfv(AL_ORIENTATION, Orientation)); }
```

Tras recompilar se obtiene una librería que, ahora sí, es capaz de posicionar correctamente sonido y oyente en 3D.

B.2.4. OpenCV, Open Source Computer Vision

(RETOMAR!!!) 2. Due to many technical problems the installation package does not include pre-compiled OpenCV libraries for Visual Studio users.

Instead, it includes libraries built with MinGW 4.3.3 TDM-SJLJ.

They are good enough to run the C/C++ and Python samples and tests, but for developing your OpenCV-based applications using Visual Studio, Borland IDE etc., or even a different version of MinGW, you need to build the libraries with your compiler using CMake, as explained here: http://opencv.willowgarage.com/wiki/InstallGuide.

Here is the procedure at glance: ———————— 1. Download CMake from http://www.cmake.org/cm and install it.

2. Run CMake GUI tool and configure OpenCV there: 2.1. select C:/OpenCV2.0 (or the installation directory you chose) as the source directory; 2.2. choose some other directory name for the generated project files, e.g. C:/OpenCV2 2.3. press Configure button, select your preferrable build environment 2.4. adjust any options at your choice 2.5. press Configure again, then press Generate. 3a. In the case of Visual Studio or any other IDE, open the generated solution/workspace/project ..., e.g. C:/OpenCV2.0/vs2008/OpenCV.sln, build it in Release and Debug configurations. 3b. In the case of command-line Makefiles, enter the destination directory and type "make"(or "nmake.^{et}c.)

4. Add the output directories to the system path, e.g.: C:/OpenCV2.0/vs2008/bin/Debug;C:/OpenCV2.0/vs2 It is safe to add both directories, since the Debug OpenCV DLLs have the "d"suffix, which the Release DLLs do not have.

5. Optionally, add C:/OpenCV2.0/include/opencv to the list of include directories in your IDE settings, and the output library directories (e.g. C:/OpenCV2.0/vs2008/lib/(Debug,Release)) to the list of library paths.

It is important to build both release and debug configurations, and link you code with the proper libraries in each configuration, otherwise various compile-time or run-time errors are possible.

ADEMAS VS 2008 EE no tiene soporte para OPENMP, así que es necesario desactivarlo en CMAKE, tambien PYTHON SUPPORT

Dependencias cv200d.lib cxcore200d.lib highgui200d.lib

B.2.5. Panda3D

(REVISAR!!!!!) www.panda3d.org/wiki/index.php/Tutorial:_Compiling_the_Panda_3D_Source_on_Winc
www.panda3d.org/wiki/index.php/How_to_build_a_CXX_Panda3D_game_using_Microsoft_Visual_Studio_20
www.panda3d.org/wiki/index.php/Configuring_Panda

scr/framework/config_famework.cxx (debug) ConfigVariableBool show_frame_rate_meter("show-frame-rate-meter", true);

panda3d usa winsock mientras que openCV usa ws2def..... sonincompatibles, debería usarse winsock2.h..

Se añade winsock2.h al comienzo de Application.h para resolver el conflicto (este problema sólo aparecería en las versiones de las librerías compliladas en Windows)

Excluir librería standar (norecuerdonombre)

Modificar fuentes de panda para exportar Geom::CDataCache y GeomVertexData::CDataCache, mediate macro EXPCL_PANDA_GOBJ : src/gobj/geom.h

// The pipelined data with each CacheEntry. class EXPCL_PANDA_GOBJ CDataCache : public CycleData public: INLINE CDataCache(); INLINE CDataCache(const CDataCache &copy); virtual CDataCache(); ALLOC_DELETED_CHAIN(CDataCache); virtual CycleData *make_copy() const; virtual TypeHandle get_parent_type() const return Geom::get_class_type();

src/gobj/geomvertexdata

class EXPCL_PANDA_GOBJ CDataCache : public CycleData public: INLINE CDataCache(); INLINE CDataCache(const CDataCache &copy); ALLOC_DELETED_CHAIN(CDataCache); virtual CycleData *make_copy() const; virtual TypeHandle get_parent_type() const return GeomVertexData::get_class_type();

cambios aceptados en revisión y subidos al repositorio

B.2.6. PostgreSQL

Inslatador de windows. Descargar instalador de Windows, ejecutar, seguir los pasos del tutorial. Instalar al mismo nivel que el proyecto. en carpeta BBDD. puerto por defecto 5432. Ejecutar pgAdmin para crear la base de datos. nombre voxDB, codificacion UTF8, tablespace pg_default, colacióntipo de caracter spanish_spain.1252. Nuevo rol de login, nombre vox, contraseña vox. Asignar usuario a base de datos.

B.2.7. Debea

set DEVEL = directorio nmake -f makefile.vc DEBUG=0 DEBUG=1 PGSQL=1 nmake -f makefile.vc DEBUG=0 DEBUG=1 PGSQL=1 install copiar directorio resultatnte en extern_debea (REVISAR!!!!!)

B.3. Interfaces core

Para preparar el módulo ha sido necesario configurar el proyecto igui de la siguiente forma:

- Additional Include Directories:

```
.
..\..\src
..\..\extern\include
```

- Additional Library Directories:

```
\"\$(OutDir)\"
```

- Preprocessor Definitions (Debug): WIN32; _DEBUG; _WINDOWS;
- Preprocessor Definitions (Release): WIN32; _WINDOWS;

Además por asuntos de compatibilidad es necesario asegurar las siguientes opciones:

- No usar la macro _NDEBUG en release.
- Configuration Type: Static Library (.lib). (actualemnte dll cambiar)
- Use of ATL: Not Using ATL.
- Common Language Runtime Support: No Common Language Runtime Support.
- Runtime Library: Multi-threaded [Debug] DLL (\MTD \MTDd).

B.3.1. IApplicationConfiguration

```
class CameraData
{ public:
   CameraData() : x(0),y(0),z(0),flip_h(false),flip_v(false){}
   ~CameraData(){}
   double x, y, z;
   bool flip_h, flip_v; };

class DisplayData
{ public:
   DisplayData() : x(0),y(0),z(0),flip_h(false),flip_v(false),
      resolution_x(800),resolution_y(600){}
   ~DisplayData() {}
   double x, y, z;
```

```
13      bool flip_h, flip_v;
14      unsigned int resolution_x, resolution_y; };
15
16  class IApplicationConfiguration
17  { public:
18      virtual ~IApplicationConfiguration(){}
19      virtual core::CameraData  GetCameraData(const int &id)=0;
20      virtual core::DisplayData GetDisplayData(const int &id)=0;
21      virtual unsigned int GetNumCams()     = 0;
22      virtual unsigned int GetNumDisplays() = 0;
23      virtual std::string GetDBHost() = 0;
24      virtual std::string GetDBPort() = 0;
25      virtual std::string GetDBName() = 0;
26      virtual std::string GetDBUser() = 0;
27      virtual std::string GetLanguage() = 0;
28      virtual std::string GetDBPassword()      = 0;
29      virtual std::string GetRootDirectory()   = 0;
30      virtual std::string GetDataDirectory()   = 0;
31      virtual std::string GetModelDirectory()  = 0;
32      virtual std::string GetSoundDirectory()  = 0;
33      virtual std::string GetImageDirectory()      = 0;
34      virtual std::string GetUIResourceDirectory() = 0;
35
36      virtual void SetRootDirectory(const std::string &value) = 0;
37      virtual void SetDataDirectory(const std::string &value) = 0;
38      virtual void SetModelDirectory(const std::string &value) = 0;
39      virtual void SetSoundDirectory(const std::string &value) = 0;
40      virtual void SetImageDirectory(const std::string &value) = 0;
41      virtual void SetUIResourceDirectory(const std::string &value) = 0;
42      virtual void SetDBHost(const std::string &value) = 0;
43      virtual void SetDBPort(const std::string &value) = 0;
44      virtual void SetDBName(const std::string &value) = 0;
45      virtual void SetDBUser(const std::string &value) = 0;
46      virtual void SetDBPassword(const std::string &value) = 0;
47      virtual void SetLanguage(const std::string &value) = 0;
48      virtual void SetNumCams(const unsigned int &value) = 0;
49      virtual void SetNumDisplays(const unsigned int &value) = 0;
50      virtual void SetCameraData(const int &id, const CameraData &value)=0;
51      virtual void SetDisplayData(const int &id, const DisplayData &value)
          =0;
52  };
```

B.4. Módulo de GUI

Para preparar el módulo ha sido necesario configurar el proyecto igui de la siguiente forma:

- Additional Include Directories:

```
.
..\..\src
..\..\extern\include
..\..\extern\include\wxwidgets
..\..\extern\include\wxwidgets\msvc
..\..\extern\boost
```

- Additional Library Directories:

```
..\..\bin
..\..\extern\lib
..\..\extern\boost\lib
\"\$(OutDir)\"
```

- Preprocessor Definitions (Debug): WIN32; _DEBUG; _WINDOWS; _USRDLL; _MSVC; _IGUIEXPORT_; __WXMSW__; __WXDEBUG__; _UNICODE; NOPCH
- Preprocessor Definitions (Release): WIN32; _WINDOWS; _USRDLL; _MSVC;_IGUIEXPORT_; __WXMSW__; _UNICODE; NOPCH
- Additional Dependencies (Debug): wxmsw29ud_core.lib wxbase29ud.lib wxtiffd.lib wxjpegd.lib wxpngd.lib wxzlibd.lib wxregexud.lib wxexpatd.lib kernel32.lib user32.lib gdi32.lib comdlg32.lib winspool.lib winmm.lib shell32.lib comctl32.lib ole32.lib oleaut32.lib uuid.lib rpcrt4.lib advapi32.lib wsock32.lib wininet.lib
- Additional Dependencies (Release): wxmsw29u_core.lib wxbase29u.lib wxtiff.lib wxjpeg.lib wxpng.lib wxzlib.lib wxregexu.lib wxexpat.lib kernel32.lib user32.lib gdi32.lib comdlg32.lib winspool.lib winmm.lib shell32.lib comctl32.lib ole32.lib oleaut32.lib rpcrt4.lib advapi32.lib wsock32.lib wininet.lib

Además por asuntos de compatibilidad es necesario asegurar las siguientes opciones:

- No usar la macro _NDEBUG en release.
- Configuration Type: Static Library (.lib).
- Use of ATL: Not Using ATL.
- Common Language Runtime Support: No Common Language Runtime Support.
- Runtime Library: Multi-threaded [Debug] DLL (\MTD \MTDd).

B.4.1. MainGui

```
#define IMPLEMENT_MIIAPP(name) IMPLEMENT_APP(name)
namespace core {namespace igui{
class MainFrame;
class MainGui : public core::IGui
{public:
        virtual ~MainGui();
        virtual void Delete();
        virtual void Update();

        void RegisterWindow(IGuiWindow *window);
        void AddWindowToFullscreenList(IGuiWindow *window);
        void ShowAll(bool value);
        void SetAllWindowsFullScreen(bool value);
        static MainGui* GetInstance(const std::wstring &title = L"");
 private:
        MainGui(const std::wstring &title = L"");
        static MainGui* instance;
        static MainFrame *main_frame;
        static std::map<IGuiWindow*, int> registered_windows;
        static std::map<IGuiWindow*, int> fullscreenable_windows;
};
}}

MainGui* MainGui::GetInstance(const std::string &title)
{       boost::mutex::scoped_lock lock(m_mutex);
        if (instance == NULL)
                instance = new MainGui(title);
        return instance; }
```

B.4.2. MainFrame

```
namespace core { namespace igui {
class MainGui;
class MainFrame : public wxFrame
{ public:
        MainFrame(wxWindow *parent, wxWindowID id, const wxString &title
           , const wxPoint &pos=wxDefaultPosition, const wxSize &size=
           wxDefaultSize, long style=wxDEFAULT_FRAME_STYLE, const
           wxString &name=wxFrameNameStr);
        virtual ~MainFrame();
        void Delete();
```

```
 private:
        wxMenu *file_menu, *view_menu, *tools_menu, *help_menu;
        wxMenuItem *item_file_close, *item_view_fullscreen, *
           item_view_start, *item_tools_configure, *item_help_about;
        wxWindow *dummy_panel;
        GUIHelp  *help_panel;
        GUIStart *start_panel;
        DECLARE_EVENT_TABLE()
        void InitShortCuts();
        void OnClose(wxCommandEvent& WXUNUSED(event));
        void OnViewStart(wxCommandEvent& WXUNUSED(event));
        void OnHelpAbout(wxCommandEvent& WXUNUSED(event));
        void DismissPanels();
};      }}
#endif
```

B.4.3. GUIStart

```
namespace core { namespace igui {
class GUIStart : public wxPanel
{ public:
        GUIStart(wxWindow *parent, wxWindowID id,  const wxPoint &pos=
           wxDefaultPosition, const wxSize &size=wxDefaultSize, long
           style=wxTAB_TRAVERSAL, const wxString &name = "panel");
        virtual ~GUIStart();
        void Delete();
        void OnPaint(wxPaintEvent &evt);
        void paintNow();
        void render(wxDC& dc);
 private:
        wxBitmap background_image;
        DECLARE_EVENT_TABLE()
        wxButton *login_button, *start_button, *configure_button;
};}}
#endif
```

B.4.4. GUIHelp

```
BEGIN_EVENT_TABLE(GUIHelp, wxPanel)
        EVT_PAINT (GUIHelp::OnPaint)
END_EVENT_TABLE()

```

```
GUIHelp::GUIHelp(wxWindow *parent, wxWindowID id, const wxPoint &pos,
   const wxSize &size, long style, const wxString &name)
: wxPanel(parent, id, pos, size, style, name)
{ background_image = wxBitmap("c:/etc/help.png", wxBITMAP_TYPE_ANY); }

GUIHelp::~GUIHelp()
{}

void GUIHelp::Delete()
{ wxPanel::Destroy(); }

void GUIHelp::OnPaint(wxPaintEvent & evt)
{ wxPaintDC dc(this);
  render(dc); }

void GUIHelp::render(wxDC& dc)
{ dc.DrawBitmap(background_image, 0, -20, false );
  dc.DrawRotatedText("ProjectVOX", 250, 30, 0);
  dc.DrawRotatedText("Code Name ::TheElectricGOAT::", 250, 50, 0);
  dc.DrawRotatedText("Version 0.1", 250, 70, 0); }
```

B.5. Módulo de Percepción

Para implementar el módulo IPercept se ha configurado el proyecto de la siguiente forma:

- Additional Include Directories:

```
.
..\..\src
..\..\extern\include
..\..\extern\include\opencv
..\..\extern\boost
..\..\extern\include\wxwidgets\msvc
..\..\extern\include\wxwidgets
..\..\extern\sfml\include
```

- Additional Library Directories:

```
..\..\bin
..\..\extern\lib
..\..\extern\lib\opencv
..\..\extern\boost\lib
..\..\extern\sfml\lib\vc2008
```

```
\"\$(OutDir)\"
```

- Preprocessor Definitions (Debug): WIN32;_DEBUG
- Preprocessor Definitions (Release): WIN32;_WINDOWS;_MSVC;
- Additional Dependencies (Debug): cv200d.lib cxcore200d.lib highgui200d.lib sfml-audio-s-d.lib sfml-system-s-d.lib
- Additional Dependencies (Release): cv200.lib cxcore200.lib highgui200.lib sfml-audio-s.lib sfml-system-s.lib

Además por asuntos de compatibilidad es necesario asegurar las siguientes opciones:

- No usar la macro _NDEBUG en release.
- Configuration Type: Static Library (.lib).
- Use of ATL: Not Using ATL.
- Common Language Runtime Support: No Common Language Runtime Support.
- Runtime Library: Multi-threaded [Debug] DLL (\MTD \MTDd).

B.5.1. MainPercept

```
#include <core/IPercept/IPercept.h>
#include <ipercept/PerceptAudio.h>
#include <ipercept/PerceptVideo.h>
#include <string>
#include <vector>
namespace core { namespace ipercept     {
 class MainPercept : public core::IPercept
 { public:
    MainPercept();
        virtual ~MainPercept();
        virtual void Delete();
        virtual void Init();
   private:
        static int num_cam;
        static PerceptAudio* perceptAudio_module;
        static PerceptVideo* perceptVideo_module;
 };} }

MainPercept::MainPercept()
```

```
{       perceptAudio_module = new PerceptAudio();
        perceptVideo_module = new PerceptVideo(); }

MainPercept::~MainPercept()
{       delete perceptAudio_module;
        delete perceptVideo_module; }

void MainPercept::Delete()
{       perceptAudio_module->Delete();
        perceptVideo_module->Delete(); }

void MainPercept::Init()
{       perceptAudio_module->Init();
        perceptVideo_module->Init(); }
```

B.5.2. PerceptAudio

```
#include <core/IPercept/IPerceptAudio.h>
#include <string>
#include <vector>
#include <boost/thread.hpp>
#include <SFML/Audio.hpp>
#include <SFML/Audio/SoundRecorder.hpp>
namespace core { namespace ipercept {
class PerceptAudio : public core::IPerceptAudio
{ public:
   PerceptAudio();
   virtual ~PerceptAudio();
   virtual void Delete();
   virtual void Init();
  private:
   static void DoInit();
   static void DoMainLoop();
   static void Iterate();
   static void Capture();

   static boost::shared_ptr<boost::thread> m_thread;
   static boost::try_mutex m_mutex;
   static bool initialized, stop_requested;
   static sf::SoundBufferRecorder Recorder;
   static sf::SoundBuffer recordingBuffer;
};}}

void PerceptAudio::Delete()
```

```
{       stop_requested = true;
        assert(m_thread);
        m_thread->join(); }

void PerceptAudio::Init()
{       PerceptAudio::DoInit(); }

void PerceptAudio::DoInit()
{       if (!initialized)
        {       assert(!m_thread);
                m_thread = boost::shared_ptr<boost::thread>(new boost::
                   thread(boost::function0<void>(&PerceptAudio::
                   DoMainLoop))); } }

void PerceptAudio::DoMainLoop()
{       initialized = true;
        Recorder.Start();
        while(!stop_requested)
        {       Iterate();
                m_thread->sleep(boost::get_system_time()+boost::
                   posix_time::milliseconds(1000)); }}

void PerceptAudio::Iterate()
{       boost::try_mutex::scoped_try_lock lock(m_mutex);
        if (lock) { Capture(); }
}

void PerceptAudio::Capture()
{       Recorder.Stop();
        recordingBuffer = Recorder.GetBuffer();
        Recorder.Start(); }
```

B.5.3. PerceptVideo

```
#include <core/IPercept/IPerceptVideo.h>
#include <ipercept/CamWindow.h>
#include <string>
#include <vector>
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
#include <boost/thread.hpp>
namespace core { namespace ipercept {
class PerceptVideo : public core::IPerceptVideo
```

```
{ public:
   PerceptVideo();
   virtual ~PerceptVideo();
   virtual void Delete();
   virtual void Init();

  private:
   static void DoInit();
   static void DoMainLoop();
   static void Iterate();
   static void Capture();

   static int num_cam;
   static std::map< int, CvCapture* > capture_cam_array;
   static std::map< std::string, CamWindow* > camWindow_array;
   static IplImage *capture_img;

   static boost::shared_ptr<boost::thread> m_thread;
   static boost::try_mutex m_mutex;
   static bool initialized, stop_requested;
};}}

PerceptVideo::PerceptVideo()
{ if (!initialized)
  { num_cam = NUM_CAM;
    for (int i=0; i<num_cam; i++)
    { std::stringstream window_name;
      window_name << "Cam" << i << ":";
      capture_cam_array[i] = cvCaptureFromCAM( i );
      camWindow_array[window_name.str()] = new CamWindow(window_name.str
         ());
} } }

PerceptVideo::~PerceptVideo()
{ boost::try_mutex::scoped_try_lock lock(m_mutex);
  if (lock)
  { for (std::map< int, CvCapture * >::iterator iter = capture_cam_array
     .begin(); iter!=capture_cam_array.end(); iter++)
    { std::stringstream window_name;
      window_name << "Cam" << iter->first << ":";
      cvReleaseCapture(&(iter->second)); }
    capture_cam_array.erase(capture_cam_array.begin(),capture_cam_array.
       begin());

    for (std::map< std::string, CamWindow* >::iterator iter =
       camWindow_array.begin(); iter!=camWindow_array.end(); iter++)
```

```
53        delete iter->second;
54      camWindow_array.erase(camWindow_array.begin(),camWindow_array.begin
          ());
55  }}
56
57  void PerceptVideo::Delete()
58  { stop_requested = true;
59    assert(m_thread);
60    m_thread->join(); }
61
62  void PerceptVideo::Init()
63  { PerceptVideo::DoInit(); }
64
65  void PerceptVideo::DoInit()
66  { if (!initialized)
67    { assert(!m_thread);
68      m_thread = boost::shared_ptr<boost::thread>(new boost::thread(boost
          ::function0<void>(&PerceptVideo::DoMainLoop)));
69  } }
70
71  void PerceptVideo::DoMainLoop()
72  { initialized = true;
73    while(!stop_requested)
74    { Iterate();
75      m_thread->sleep(boost::get_system_time()+boost::posix_time::
          milliseconds(10));
76  } }
77
78  void PerceptVideo::Iterate()
79  {  boost::try_mutex::scoped_try_lock lock(m_mutex);
80     if (lock) { Capture(); } }
81
82  void PerceptVideo::Capture()
83  { for (std::map<int, CvCapture *>::iterator iter = capture_cam_array.
       begin(); iter!=capture_cam_array.end(); iter++)
84    { std::stringstream window_name;
85      window_name << "Cam" << iter->first << ":";
86      capture_img = cvQueryFrame(iter->second);
87      std::map< std::string, CamWindow* >::iterator cam_iter =
          camWindow_array.find(window_name.str());
88      if (cam_iter != camWindow_array.end())
89        cam_iter->second->ShowImage(capture_img);
90  } }
```

B.6. Módulo de Producción

Para implementar el módulo IProd se ha configurado el proyecto de la siguiente forma:

- Additional Include Directories:

```
.
..\..\src
..\..\extern\include
..\..\extern\panda3d\built\include
..\..\extern\panda3d\built\python\include
..\..\extern\boost
..\..\extern\include\wxwidgets\msvc
..\..\extern\include\wxwidgets
..\..\extern\sfml\include
```

- Additional Library Directories (Debug):

```
..\..\bin
..\..\extern\lib
..\..\extern\panda3d\debug\lib
..\..\extern\panda3d\debug\python
..\..\extern\panda3d\debug\python\Lib
..\..\extern\panda3d\debug\python\libs
..\..\extern\boost\lib
..\..\extern\sfml\lib\vc2008
\$(OutDir)
```

- Additional Library Directories (Release):

```
..\..\bin
..\..\extern\lib
..\..\extern\panda3d\built\lib
..\..\extern\panda3d\built\python
..\..\extern\panda3d\built\python\Lib
..\..\extern\panda3d\built\python\libs
..\..\extern\boost\lib
..\..\extern\sfml\lib\vc2008
\$(OutDir)
```

- Preprocessor Definitions (Debug): WIN32;_DEBUG
- Preprocessor Definitions (Release): WIN32;

- Additional Dependencies (Debug): libpandaexpress.lib libp3framework.lib libpanda.lib libpandafx.lib libp3dtool.lib libp3dtoolconfig.lib libp3pystub.lib libp3direct.lib sfml-audio-s-d.lib sfml-system-s-d.lib
- Additional Dependencies (Release): libpandaexpress.lib libp3framework.lib libpanda.lib libpandafx.lib libp3dtool.lib libp3dtoolconfig.lib libp3pystub.lib libp3direct.lib sfml-audio-s.lib sfml-system-s.lib

Además por asuntos de compatibilidad es necesario asegurar las siguientes opciones:

- No usar la macro _NDEBUG en release.
- Configuration Type: Static Library (.lib).
- Use of ATL: Not Using ATL.
- Common Language Runtime Support: No Common Language Runtime Support.
- Runtime Library: Multi-threaded [Debug] DLL (\MTD \MTDd).

B.6.1. Prod3dWindow

```
#include <igui/maingui.h>
#include <string>
#include <vector>
#include <pandaFramework.h>
namespace core { namespace iprod {
class Prod3DWindow : public core::IGuiWindow
{ public:
   Prod3DWindow(PandaFramework *framework, const WindowProperties &props
      , bool registrable = false, bool fullscreenable = false);
   virtual ~Prod3DWindow();
   WindowFramework *GetWindowFrameWork() {return m_windowFramework;}
   void Show(const bool &value = true);
   void FullScreen( const bool &value = true );
   bool IsShown() {return isShown;}
 private:
   bool isShown;
   WindowFramework *m_windowFramework;
}; }}
```

B.6.2. MainProd

```
#include <core/IProd/IProd.h>
#include <iprod/Prod3DWindow.h>
#include <string>
#include <pandaFramework.h>
#include <pandaSystem.h>
#include <modelPool.h>
#include <filename.h>
#include <genericAsyncTask.h>
#include <asyncTaskManager.h>
#include "cIntervalManager.h"
#include "cLerpNodePathInterval.h"
#include "cMetaInterval.h"
#include <boost/thread.hpp>
#include <SFML/Audio.hpp>

namespace core { namespace iprod {
class MainProd : public core::IProd
{ public:
   MainProd(int argc, char *argv[]);
   virtual ~MainProd();
   virtual void Delete();
   virtual void Init();
   virtual void DoStuff();
   virtual void OpenWindow();
  private:
   static void DoInit();
   static void DoDoStuff();
   static void PlaySoundCapture();
   static void DoMainLoop();
   static void Iterate();
   static void CreateDefaultWindows(int numWindows);
   static void LoadDefaultScene();

   static AsyncTask::DoneStatus SpinCameraTask(GenericAsyncTask* task,
      void* data);
   static boost::shared_ptr<boost::thread> m_thread;
   static boost::try_mutex m_mutex;
   static PandaFramework framework;
   static Thread *graphic_thread;
   static PT(AsyncTaskManager) taskMgr;
   static PT(ClockObject) globalClock;
   static bool initialized, stop_requested;
   static std::map<int, Prod3DWindow*>          prod3Dwindow_array;
   static std::map<int, WindowFramework*>       pandawindows_array;
```

```
44      static std::map<int, NodePath>
           windowcamera_array;
45      static int last_window_id, m_argc;
46      static char **m_argv;
47      static NodePath cam_viewpoint;
48      static NodePath origin, up;
49
50      static sf::SoundBuffer sound_Buffer;
51      static double listener_position[];
52      static double listener_target[];
53      static double sound_pos[];
54      static sf::Sound Sound;
55  }; } }
56
57  MainProd::MainProd(int argc, char *argv[])
58  { m_argc = argc;
59    m_argv = argv; }
60
61  MainProd::~MainProd()
62  { m_thread->join();
63    for (std::map<int, Prod3DWindow*>::iterator iter=prod3Dwindow_array.
         begin(); iter != prod3Dwindow_array.end(); iter++)
64     delete iter->second;
65    prod3Dwindow_array.erase(prod3Dwindow_array.begin(),
         prod3Dwindow_array.end());
66  }
67
68  void MainProd::Delete()
69  { stop_requested = true;
70    assert(m_thread);
71    m_thread->join(); }
72
73  void MainProd::DoMainLoop()
74  { framework.open_framework( m_argc, m_argv);
75    taskMgr = AsyncTaskManager::get_global_ptr();
76    globalClock = ClockObject::get_global_clock();
77    graphic_thread = Thread::get_current_thread();
78
79    CreateDefaultWindows(DEFAULT_NUM_WINDOWS);
80
81    LoadDefaultScene();
82
83    sf::SoundBuffer Buffer;
84    if (!Buffer.LoadFromFile("c://etc//motor.wav"))
85     int error_loadfromfile = 1;
86    unsigned int chan = Buffer.GetChannelsCount();
```

```
87    sf::Listener::SetGlobalVolume(100.f);
88    Sound.SetBuffer(Buffer);
89    Sound.SetLoop(true);
90    Sound.SetPosition(sound_pos[0], sound_pos[1], sound_pos[2]);
91    Sound.SetMinDistance(10.f);
92    Sound.SetAttenuation(0.75f);
93
94    initialized = true;
95
96    Sound.Play();
97
98    while(!stop_requested)
99    { Iterate();
100     m_thread->sleep(boost::get_system_time()+boost::posix_time::
          milliseconds(10)); }
101
102   framework.close_all_windows();
103   framework.close_framework();
104 }
105
106 void MainProd::Iterate()
107 { boost::try_mutex::scoped_try_lock lock(m_mutex);
108   if (lock)
109   { framework.do_frame(graphic_thread);
110     CIntervalManager::get_global_ptr()->step(); }
111 }
112
113 void MainProd::Init()
114 { MainProd::DoInit(); }
115
116 void MainProd::DoStuff()
117 { MainProd::DoDoStuff(); }
118
119 void MainProd::DoInit()
120 { if (!initialized)
121         { assert(!m_thread);
122       m_thread = boost::shared_ptr<boost::thread>(new boost::thread(
          boost::function0<void>(&MainProd::DoMainLoop))); }
123 }
124
125 void MainProd::DoDoStuff()
126 { boost::try_mutex::scoped_try_lock lock(m_mutex);
127   if (lock && initialized)
128   { double time = globalClock->get_real_time();
129     double angledegrees = time * 2.0;
130     double angleradians = angledegrees * (3.14 / 180.0);
```

```
131      windowcamera_array[1].set_pos(20*sin(angleradians),-20.0*cos(
             angleradians),3);
132      windowcamera_array[1].set_hpr(angledegrees, 0, 0);
133
134      LPoint3f cam_pos = windowcamera_array[2].get_pos();
135      LVector3f abs_vec_at, abs_vec_up;
136      abs_vec_at = cam_viewpoint.get_pos(pandawindows_array[2]->get_render
             ());
137      abs_vec_up = up.get_pos(pandawindows_array[2]->get_render());
138      LPoint3f new_pos(cam_pos.get_x()  ,    cam_pos.get_z(),    -cam_pos.
             get_y());
139      LPoint3f new_at(abs_vec_at.get_x(), abs_vec_at.get_z(), -abs_vec_at.
             get_y());
140      LPoint3f new_up(abs_vec_up.get_x(), abs_vec_up.get_z(), -abs_vec_up.
             get_y());
141      new_at = new_at - new_pos;
142      new_up = new_up - new_pos;
143      new_at.normalize();
144      new_up.normalize();
145      sf::Listener::SetPosition(new_pos.get_x(), new_pos.get_y(), new_pos.
             get_z());
146      sf::Listener::SetTarget(new_at.get_x(), new_at.get_y(), new_at.get_z
             () , new_up.get_x(), new_up.get_y(), new_up.get_z());
147     }
148     else
149     { //más suerte la próxima }
150  }
151
152  void MainProd::CreateDefaultWindows(int num_windows)
153  { WindowProperties win_props = WindowProperties();
154    win_props.set_size(800, 600);
155    for (int i=last_window_id+1; i <= last_window_id + num_windows; i++)
156    { prod3Dwindow_array[i] = new Prod3DWindow(&framework, win_props, true
         , true);
157      pandawindows_array[i] = prod3Dwindow_array[i]->GetWindowFrameWork();
158      pandawindows_array[i]->set_background_type(WindowFramework::
             BackgroundType::BT_white);
159      pandawindows_array[i]->set_lighting(true);
160      pandawindows_array[i]->set_perpixel(true);
161      windowcamera_array[i] = pandawindows_array[i]->get_camera_group();
162    }
163    origin = pandawindows_array[1]->load_model(framework.get_models(), "
         panda-model");
164    origin.set_pos(0,0,0);
165    origin.set_scale(0.001);
166    origin.reparent_to(pandawindows_array[2]->get_render());
```

```
167     up = pandawindows_array[1]->load_model(framework.get_models(), "panda-
          model");
168     up.set_pos(0,0,10);
169     up.set_scale(0.002);
170     up.reparent_to(pandawindows_array[2]->get_camera_group());
171     cam_viewpoint = pandawindows_array[1]->load_model(framework.get_models
          (), "panda-model");
172     cam_viewpoint.set_pos(0,10,0);
173     cam_viewpoint.set_scale(0.002);
174     cam_viewpoint.reparent_to(pandawindows_array[2]->get_camera_group());
175     last_window_id += num_windows;
176   }
177
178   void MainProd::LoadDefaultScene()
179   { if (pandawindows_array.begin() != pandawindows_array.end())
180     { NodePath environment = pandawindows_array[1]->load_model(framework.
          get_models(),"environment");
181       environment.set_scale(0.25,0.25,0.25);
182       environment.set_pos(-8,42,0);
183       environment.reparent_to(pandawindows_array[1]->get_render());
184       NodePath pandaActor = pandawindows_array[1]->load_model(framework.
            get_models(), "panda-model");
185       pandaActor.set_scale(0.005);
186       LPoint3f pandapos = pandaActor.get_pos();
187       pandaActor.reparent_to(pandawindows_array[1]->get_render());
188       NodePath pandaActor2 = pandawindows_array[1]->load_model(framework.
            get_models(), "panda-model");
189       pandaActor2.set_scale(0.004);
190       pandaActor2.set_pos(10.0,0.0,0.0);
191       LPoint3f pandapos2 = pandaActor2.get_pos();
192       pandaActor2.reparent_to(pandawindows_array[1]->get_render());
193           ...
194
195       pandawindows_array[1]->load_model(pandaActor, "panda-walk4");
196       pandawindows_array[1]->loop_animations(0);
197
198       std::map<int, WindowFramework*>::iterator iter = pandawindows_array.
            begin();
199       iter++;
200       while(iter != pandawindows_array.end())
201       { environment.instance_to(iter->second->get_render());
202         pandaActor.instance_to(iter->second->get_render());
203         pandaActor2.instance_to(iter->second->get_render());
204         pandaActor3.instance_to(iter->second->get_render());
205         pandaActor4.instance_to(iter->second->get_render());
206         iter->second->setup_trackball();
```

```
207         iter++;
208 } } }
```

B.7. Aplicación Principal

El proyecto que conforma la aplicación princial se ha configurado de la siguiente forma:

- Additional Include Directories:

```
1  .
2  ..\..\src
3  ..\..\extern\include
4  ..\..\extern\panda3d\built\include
5  ..\..\extern\panda3d\built\python\include
6  ..\..\extern\include\opencv
7  ..\..\extern\include\wxwidgets
8  ..\..\extern\include\wxwidgets\msvc
9  ..\..\extern\boost
10 ..\..\extern\sfml\include
```

- Additional Library Directories (Debug):

```
1 ..\..\bin
2 ..\..\extern\lib
3 ..\..\extern\bin
4 ..\..\extern\panda3d\debug\python\libs
5 ..\..\extern\boost\lib
6 \$(OutDir)
```

- Additional Library Directories (Release):

```
1 ..\..\bin
2 ..\..\extern\lib
3 ..\..\extern\bin
4 ..\..\extern\panda3d\built\python\libs
5 ..\..\extern\boost\lib
6 \$(OutDir)
```

- Preprocessor Definitions (Debug): WIN32;_DEBUG;_WINDOWS;_MSVC;
- Preprocessor Definitions (Release): WIN32;_WINDOWS;_MSVC;

- Additional Dependencies (Debug): icogd.lib iguid.lib iperceptd.lib ipersistenced.lib iprodd.lib
- Additional Dependencies (Release): icog.lib igui.lib ipercept.lib ipersistence.lib iprod.lib
- Environment (Debug):

```
PATH=%PATH%;..\..\extern\bin;..\..\extern\bin\opencv;..\..\extern\panda3d\debug\bin;..\..\
    extern\panda3d\debug\python;..\..\extern\panda3d\debug\python\DLLs;..\..\extern\sfml\
    extlibs\bin;
```

- Environment (Release):

```
PATH=%PATH%;..\..\extern\bin;..\..\extern\bin\opencv;..\..\extern\panda3d\built\bin;..\..\
    extern\panda3d\built\python;..\..\extern\panda3d\built\python\DLLs;..\..\extern\sfml\
    extlibs\bin;
```

Además por asuntos de compatibilidad es necesario asegurar las siguientes opciones:

- No usar la macro _NDEBUG en release.
- Configuration Type: Application (.exe).
- Use of ATL: Not Using ATL.
- Common Language Runtime Support: No Common Language Runtime Support.
- Runtime Library: Multi-threaded [Debug] DLL (\MD \MDd).

B.7.1. Application

```
1  #ifdef _WINDOWS
2  #include <winsock2.h>
3  #endif
4  #include <core/IApplication.h>
5  #include <core/IGUI/IGui.h>
6  #include <core/ICog/ICog.h>
7  #include <core/IProd/IProd.h>
8  #include <core/IPercept/IPercept.h>
9  #include <core/IPersistence/IPersistence.h>
10 #include <igui/maingui.h>
11 #include <icog/maincog.h>
12 #include <iprod/mainprod.h>
13 #include <ipercept/mainpercept.h>
14 #include <ipersistence/mainpersistence.h>
15 class Application : public wxApp, public core::IApplication
16 { public:
```

```
17      Application(void);
18      ~Application(void);
19      bool OnInit();
20      void ExitApplication();
21    private:
22      DECLARE_EVENT_TABLE();
23      wxTimer app_timer;
24      core::IGui           *app_maingui;
25      core::IPercept       *app_mainpercept;
26      core::IProd          *app_mainprod;
27      void OnIdle(wxIdleEvent &event);
28      int          OnRun();
29      void DoMainLoopStuff(wxTimerEvent& event);
30  };
31
32  #define MAINLOOP_EVT 12345
33
34  BEGIN_EVENT_TABLE(Application, wxApp)
35          EVT_IDLE        (Application::OnIdle)
36          EVT_TIMER       (MAINLOOP_EVT, Application::DoMainLoopStuff)
37  END_EVENT_TABLE()
38
39  Application::Application(void) : app_maingui(NULL), app_mainpercept(NULL
       ), app_mainprod(NULL)
40  { app_timer.SetOwner(this, MAINLOOP_EVT); }
41
42  Application::~Application(void)
43  {       if (app_mainpercept!=NULL)
44          {       app_mainpercept->Delete();
45                  delete app_mainpercept; }
46
47          if (app_mainprod!=NULL)
48          {       app_mainprod->Delete();
49                  delete app_mainprod;    }
50
51          if (app_maingui!=NULL)
52          {       app_maingui->Delete();
53                  delete app_maingui;     } }
54
55  bool Application::OnInit()
56  { app_maingui     = core::igui::MainGui::GetInstance("VOX");
57    app_mainpercept = (core::IPercept *) new core::ipercept::MainPercept()
        ;
58    app_mainprod    = (core::IProd *) new core::iprod::MainProd(argc, argv
        );
59    app_mainpercept->Init();
```

```
60    app_mainprod->Init();
61    return true; }
62
63  void Application::OnIdle(wxIdleEvent &event)
64  { //To Do less important/frequent stuff }
65
66  int Application::OnRun()
67  { app_timer.Start(10);
68    return wxApp::OnRun(); }
69
70  void Application::ExitApplication()
71  {       wxApp::Exit(); }
72
73  void Application::DoMainLoopStuff(wxTimerEvent& event)
74  { //To Do important/frequent stuff
75    app_mainprod->DoStuff(); }
```

B.7.2. main

```
1  #define _WINSOCKAPI_
2  #include "Application.h"
3  IMPLEMENT_MIIAPP(Application)
```

Apéndice C

Comentarios Generales sobre la redacción de los proyectos

Por donde se empieza? Habitualmente se empieza a redactar en orden inverso al que se lee el documento, primero se puede empezar a rellenar la bibliografía utilizada que se irá completando según avance el proyecto. A continuación, a partir de un esqueleto inicial del proyecto, que puede estar redactado, manuscrito, o simplemente en la cabeza del estudiante, se empieza a redactar las partes más concretas del proyecto, que tengamos más claras, y que sean lo más independiente posible de la redacción de otras partes del proyecto. Por ejemplo, se puede empezar por anexos donde se resuma las características de una herramienta que utilizamos, etc.., a continuación empezamos a redactar de manera individual los detalles de cada una de las etapas en las que se constituye el proyecto, no tienen que redactarse ordenadas según aparecen en el texto, sobre la elección sobre cual empezar, siempre primará que sea una parte que tengamos bien clara, y que hayamos delimitado su contenido para que sea independiente de la redacción de las otras etapas. Como verán, según vayan avanzando en la redacción, cada vez verán las cosas más claras, y de forma natural verán la forma de ir redactando las otras partes del proyecto, hasta llegar a las secciones de introducción y conclusiones y resultados que son las más delicadas de desarrollar, pues son las más importantes y las que previsiblemente se van a leer en mayor detalle las personas que lean el proyecto. Una deficiente redacción de la introducción (que es donde se atrae al lector sobre la importancia de lo que se va a hacer) o una mala presentación de las conclusiones y resultados (que es donde se transmite el mensaje de todo lo bueno que hay en proyecto) pone en entredicho la calidad global del proyecto. Una buena estrategia consiste en según se van redactando las diferentes secciones del proyecto, ir haciendo un borrador de las secciones de introducción y conclusiones y resultados, poniendo las ideas sueltas, y en principio desordenadas, que nos vayan surgiendo y que puedan ser de utilidad en la redacción de estas secciones. Redactar bien tiene su dificultad y no todos los días tenemos la inspiración adecuada, para esos días negros, que no nos viene nada a la cabeza, lo mejor es dedicarse a cosas más mecánicas que no requieren tanta concentración, como puede ser completar la bibliografía, ir haciendo un manual de usuario o un anexo técnico, etc..

Cuando empezamos a redactar, siempre es necesario tener en cuenta algunos criterios básicos como son:

1. Escribir de cada cosa su esencia. Que es lo que es realmente relevante en la sección que estoy

redactando y esforzarme en que ello quede claro

2. Ponerse en el lugar del potencial lector. El orden y la forma en la redactamos no es sólo para que nosotros tengamos claro lo que hacemos, es sobre todo para que una tercera persona que lea el texto lo pueda tener, si cabe, más claro que nosotros. Para ello hay que respetar un orden lógico en la forma en que presentamos las cosas y no presuponer que el lector conoce los entresijos de lo que estamos haciendo, hay que evitar dar saltos en el vacío, por ejemplo dando por supuesto conocimientos que el lector no tiene o alterando el orden natural en que deben aparecer las cosas.

Algunas ideas sueltas sacadas del libro Çomo elaborar y presentar un trabajo escritoçuyo autor es el profesor Santos Pérez:

El proyecto fin de carrera es un trabajo personal en el que el estudiante debe demostrar que domina el tema, sabe organizarlo, estructurarlo y elaborar en profundidad, y presentarlo en la forma normalizada de un trabajo técnico o científico. Es la ocasión que tiene el estudiante de demostrar que sabe analizar un problema, sabe seleccionar la metodología y técnicas apropiadas para reunir los datos, y alcanzar conclusiones razonables. Un proyecto de esta naturaleza permite la evaluación de la capacidad del estudiante para aplicar su conocimiento a un tema concreto.

El proyecto fin de carrera debe ser

- Proyecto Personal: debe ser un producto de la reflexión, investigación y esfuerzo del estudiante. Si se hace con reflexión, con investigación y esfuerzo personal, el rendimiento que obtiene el estudiante es muy productivo y beneficioso para él y de una duración permanente.

- Es un trabajo documentado: es decir, serio, científico, hay que sustentar las afirmaciones con datos comprobables y lógicamente fundados.

- Planificado: La elaboración de un proyecto fin de carrera es un proceso complicado. Un trabajo de esta naturaleza requiere una planificación cuidadosa del tiempo: tiempo para investigar y documentarse, tiempo para reflexionar, tiempo para corregir posibles desviaciones y finalmente tiempo para redactarlo y presentarlo de forma adecuada.

El esquema final de un proyecto fin de carrera debe llenar las siguientes características:

- Claridad: la claridad se consigue sobre todo con una nítida división y distribución del esquema. Y a su vez esta claridad deriva también de la compresión en profundidad del material recogido.

- Convergencia hacia el objetivo: El secreto de la claridad está en saber ordenar las partes del trabajo hacia el objetivo buscado; es decir, en lograr que cada punto del esquema nos vaya encaminando con naturalidad hacia la meta enunciada en el título del trabajo.

- Coherencia: las distintas partes, puntos o párrafos deben estar trabados entre sí, concatenados, de forma que se vayan preparando y completandose recíprocamente para conseguir el efectode que cada punto sea consecuencia del otro, formando un todo orgánico y no una mera yuxtaposición de partes; por el contrario que se vaya mostrando la conexión y la coherencia lógica de los distintos aspectos tratados

- Conformidad con el objetivo: La estructura del esquema final debe resaltar lo más importante y debe dejar en la penumbra los accesorios.
- Elegancia: en la distribución del esquema, debe guardarse una cierta simetría y proporción. La elegancia no debe subordinarse a la claridad y a la verdad; pero hay una elegancia no sólo formal, sino de concepción y elegancia que contribuye significativamente a conseguir la armonía y transparencia en la transmisión del contenido principal del tema. Conviene resaltar esta elegancia sobre todo ahora que nos encontramos en un mundo de zafiedad y donde se hace gala del caos mental como norma de actuación.
- El descanso inteligente: Una vez que se ha acabado la primera redacción del proyecto se sugiere tomarse unos días de descanso suficientes para que la cabeza descanse del tema. Con este descanso se adquiere perspectiva, y aumenta la objetividad y sentido crítico del autor.

Bibliografía

[1] Leslie Lamport *LaTex : A document Preparation System.* Addison-Wesley, 1986.

[2] Christian Rolland *LaTex guide pratique.* Addison-Wesley, 1993.

[3] M. Castrillón Santana, C. Guerra Artal and M. Hernández Tejera *Real-time Detection of Faces in Video.* Face Processing in Video 2005, Victoria, Canada.

[4] R. Chellappa et al. *Human and machine recognition of faces: A survey.* Proceedings IEEE, vol. 83(5), 705-740, 1995.

[5] G. Cielniak, M. Miladinovic, D. Hammarin, L. Göransson, A. Lilienthal and T. Duckett *Appearance-based Tracking of Persons with an Omnidirectional Vision Sensor* Proceedings of the Fourth IEEE Workshop on Omnidirectional Vision (Omnivis 2003)", Madison, Wisconsin", 2003

[6] O. Déniz, A. Falcón, J. Méndez, M. Castrillón *Useful Computer Vision Techniques for Human-Robot Interaction.* International Conference on Image Analysis and Recognition, September 2004, Porto, Portugal.

[7] E. Hjelmas y B. K. Low *Face Detection: A Survey.* Computer Vision and Image Understanding, vol. 83(3), 2001.

[8] José Iges *El espacio. El tiempo en la mirada del sonido.* Catálogo de exposición. Kulturanea. España, 1999.

[9] Myron W. Krueger, Thomas Gionfriddo y Katrin Hinrichsen *VIDEOPLACE: An Artificial Reality* Proceedings of the SIGCHI conference on Human factors in computing systems, 35-40, 1985.

[10] Golan Levin y Zachary Lieberman *In-Situ Speech Visualization in Real-Time Interactive Installation and Performance.* The 3rd International Symposium on Non-Photorealistic Animation and Rendering (NPAR) June 7-9 2004, Annecy, France

[11] A. Samal and P. A. Iyengar *Automatic Recognition and Analysis of Human Faces and Facial Expressions: A Survey.* Pattern Recognition, vol. 25(1), 1992.

[12] Anne Morgan Spalter *The Computer in The Visual Arts.* Addison-Wesley Professional. 1st edition, 1999.

[13] P. Viola and M. J. Jones *Rapid Object Detection using a Boosted Cascade of Simple Features.* In Computer Vision and Pattern Recognition, 2001a.

[14] M. H. Yang et al. *Detecting Faces in Images: A Survey.* Transactions on Pattern Analysis and Machine Intelligence, vol. 24(1), 34-58, 2002.

[15] Willow Garage: S. Hassan, S. Cousins, B. Gerkey et al. *OpenCV, Open Source Computer Vision* Official Page and Documentation: http://opencv.willowgarage.com/documentation/index.html 2009.

[16] G. Wang, P. Cook et al. *Chuck, Strongly-timed, Concurrent, and On-the-fly Audio Programming Language* http://chuck.cs.princeton.edu/ 2009.

[17] Devmasters.net *3D Engines Data Base* http://www.devmaster.net/engines/ 2009

[18] Epic Games *Unreal Development Kit* http://www.udk.com/ 2009

[19] Epic Games *Unreal Script Language Reference, Example Program Structure* http://udn.epicgames.com/Three/UnrealScriptReference.html/ 2009

[20] Epic Games *Unreal Engine 3: Rendering Feature List* http://www.unrealtechnology.com/features.php?ref=rendering 2009

[21] CryTek *CryENGINE 3 Educational SDK License* http://mycryengine.com/index.php?conid=42 2009

[22] CryTek *CryENGINE 3 - Specifications* http://www.crytek.com/technology/cryengine-3/specifications/ 2009

[23] Crytek *CryTek Official Modding Portal* Educational community area for the CryENGINE 3 Software Development Kit: http://www.crymod.com/ 2009

[24] wxWidgets *wxWidgets, Cross platform GUI library* http://www.wxwidgets.org/ 2009

[25] wxWidgets *wxWidgets Features* http://www.wxwidgets.org/about/feature2.htm 2009

[26] K. Beck, J. Sutherland et al. *Manifesto for Agile Software Development* http://www.agilemanifesto.org 2001

[27] R. Colusso *Desarrollo ágil de software* http://knol.google.com/k/desarrollo-ágil-de-software 2009

[28] I. Jacobson Introducing the Essential Unified Process 2009

[29] D. Abrahams et al. Boost C++ Liraries http://www.boost.org/ 2010

[30] L. Gomila Simple and Fast Multimedia Library http://www.sfml-dev.org 2010

[31] PostGreSQL *PostGreSQL, The world most advanced open source database* http://www.postgresql.org/about/ 2010

[32] Debea *Debea Database Access Library* http://www.debea.net/ 2010